UNIVERSITAT POLITÈCNICA DE CATALUNYA

Monitoratge de suspensió, gas i inclinació d'una supermotard de competició

10 de juny de 2016

Memòria del projecte que presenta Pol Rodoreda Valeri
sota la direcció del Dr. Eng. Jordi Bonet
i la codirecció de l'Eng. Alexis López
per assolir el grau d'Enginyer en Sistemes TIC.

Al Jordi i l'Alexis, per l'ajuda i la paciència durant aquest temps.
Al Jaume Ferrer, per la col·laboració.
A l'Àngel Bertrans, per la construcció del prototip.
A la família, per ser-hi present i donar suport moral.
A TOTS, MOLTES GRÀCIES!

Índex

Índex de figures

Índex de taules

Abstract

Nowadays and under big economic investment which is having a monitoring kit, this project deals with the monitoring of suspension, gas and inclination of a racing supermotard which participating in the Spanish Supermotard Championship organized by *Real Federación Motociclista Española* (RFME).

The system consists of a functional prototype that saves data read from different sensors in a microSD card for further computational treatment offering a clear view, ideal for professional interpretation, in order to adjust certain mechanical and driving parameters that can help the pilot improve his times in each race.

To ensure the components and facilitate the placement of the prototype, has been designed a metallic encapsulated with some holes for connecting all the components. The project has also been equipped with autonomy including an external 9V battery which has sufficient capacity to maintain all of it active for hours.

Finally, to facilitate data representation, control step back and draw the shape of the circuits, has been incorporate to the prototype a GPS module.

Resum

En els temps actuals i sota la gran inversió econòmica que suposa disposar d'un equip de monitoratge, el present projecte s'ocupa del monitoratge de suspensió, gas i inclinació d'una supermotard de competició que participa al Campionat d'Espanya de Supermotards que organitza la *Real Federación Motociclista Española* (RFME).

El sistema consisteix en un prototip funcional que emmagatzema les dades que llegeix de diferents sensors en una targeta microSD per a un posterior tractament computacional oferint-ne una clara visualització, ideal per la interpretació professional, amb la finalitat d'ajustar certs paràmetres mecànics i de pilotatge que puguin ajudar al pilot a millorar els seus temps a cada cursa.

Per tal d'assegurar els components i facilitar la col·locació del prototip, s'ha dissenyat un encapsulat mecànic amb alguns orificis per a la connexió de tots els components. També s'ha dotat el projecte d'autonomia connectant-ho tot plegat a una bateria externa de 9V amb capacitat suficient per mantenir-ho activat durant hores.

Finalment per facilitar la representació de les dades, controlar el pas per volta i dibuixar la forma dels circuits, s'ha incorporat al prototip un mòdul GPS.

Part I.

Memòria

1. Introducció

Abans de començar a entrar en detall sobre el treball final de grau i veure'n els tecnicismes que ens permetran concloure amb el problema a resoldre, cal deixar clar de forma senzilla i adequada què és un sistema de monitoratge i quina n'és la seva funció.

Segons l'enciclopèdia catalana, monitoratge és un sistema d'observació, mesurament o avaluació d'un fenomen o procés, a fi de poder realitzar, en cas necessari, intervencions correctores. Per tant, en altres paraules i extrapolat al projecte que ens ocupa, monitoritzar suposa guardar dades de diferents paràmetres al llarg d'un cert temps, per a un posterior anàlisi.

És molt important no confondre un sistema de monitoratge amb un sistema de telemetria. La diferència principal entre ambdós sistemes és la forma de transmetre les dades. En el primer cas, les dades solen emmagatzemar-se en una memòria (interna o extraïble) que pertany al propi dispositiu. Per altra banda, la telemetria permet el mesurament a distància a través d'un enllaç de comunicacions sense fils. Com a curiositat, citar que la Federació Internacional de Motociclisme no permet l'ús de telemetria si aquesta envia dades a temps real al centre de control.

Aquest treball final de grau recull la teoria, pràctica i experiència de dissenyar un sistema de monitoratge que permetrà l'ajust de pilotatge i de components mecànics d'una supermotard amb la finalitat de millorar-ne el rendiment i obtenir així millors resultats a través de les possibles millores a realitzar un cop avaluades les dades.

El sistema en qüestió està dotat d'un conjunt de sensors i un mòdul GPS, connectats a una centraleta encarregada de la lectura d'aquests i l'emmagatzematge de les dades. Finalment, a través de software, es farà les gràfiques de les dades per permetre'n una visualització i anàlisi complet.

1.1. Estat de l'art

Actualment els sistemes de monitoratge són presents en àmbits com la indústria, l'aeronàutica, les plantes químiques, etc. En aquest treball ens centrarem en els sistemes de monitoratge aplicats al món del motor, especialment al motociclisme.

Actualment els sistemes de monitoratge són propis de l'alta competició. En el cas que ens ocupa, el del motociclisme, parlem de categories com Moto2 i MotoGP; equips amb pressupostos elevats que es poden permetre comprar kits sencers (figura 1.1) on, el més bàsic, incorpora sensor lambda[1], sensor de suspensió (davanter i posterior), sensor de frenada, un datalogger amb pantalla i el software necessari per la visualització i estudi de les dades. Aquest kit es pot

[1]Sensor situat al conducte d'escapament de forma que permet la mesura de concentració d'oxigen als gasos d'escapament abans que pateixin alguna alteració.

adquirir de la marca *2d-datarecording* [2dd16b] i té un preu d'uns 6.730,00€. Tots els càlculs que es realitzen per extreure dades vàlides, estan ocults dins el software privat que proporciona la pròpia marca.

Pel que fa a la inclinació, aquest és un paràmetre que només és present a la competició de motoGP donada la complexitat de càlcul i cost del sensor. Per fer-nos una idea dels preus que tracten aquestes elits, un sensor que mesura velocitat angular i acceleració de la marca *AIM Sportline* té un cost de 176,00€. Òbviament cal remarcar que l'usuari final no té cap mena de poder sobre el sensor i les dades que ofereix; simplement connectar i veure'n els resultats a través del programa de visualització i anàlisi de dades.

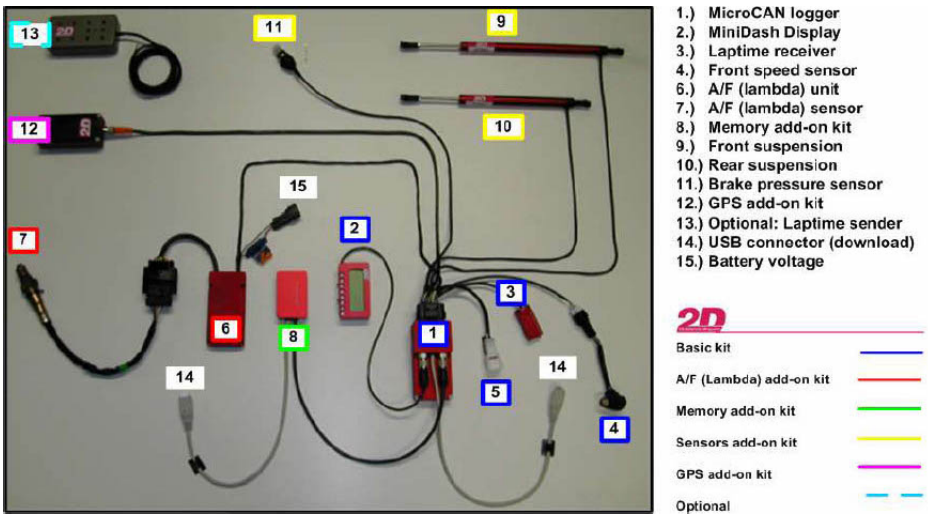


Figura 1.1.: Kit de monitoratge de competició

1.2. Motivació

Abans de descriure els objectius que ha de complir aquest treball final de grau, m'agradaria introduir d'on va sorgir la idea i la motivació de tirar-lo endavant.

Després de proposar varis temes al Departament de Disseny i Programació de Sistemes Electrònics (DIPSE) de l'Escola Politècnica Superior d'Enginyeria de Manresa (EPSEM), un bon dia el codirector del treball, l'enginyer Alexis López, em va suggerir el projecte de dissenyar un sistema de monitoratge d'una supermotard. La idea va arribar-li a través d'un familiar del propietari de la moto, el senyor Jaume Ferrer. Aquest, com a pilot particular del Campionat d'Espanya de Supermotard, no disposa del pressupost d'una escuderia i li és molt complicat adquirir un sistema de monitoratge com els descrits anteriorment.

Pensar en el fet de poder desenvolupar un producte i posar-lo a la pràctica en un àmbit real, va ser el primer impuls per tirar endavant. Per altra banda, des de fa ja uns anys que el món del motociclisme em crida l'atenció; especialment el món de l'enduro; però, davant una proposta com aquesta, quin fanàtic de les motos ho refusaria?

1.3. Objectius

Seguidament ens vam posar en contacte amb el Jaume per establir les bases del projecte. Ens va ensenyar la moto amb la qual disputa el mundial, una Yamaha YZ450F (figura 1.2). És un model de motocròs, però està modificada per complir els requisits necessaris que s'exigeixen per considerar-la supermotard. Ens va plantejar les idees que tenia i va comentar que havia treballat de mecànic en una escuderia que utilitzava sistema de monitoratge; per tant, ens va poder ajudar sobre la presa de decisions inicials.

L'objectiu principal del projecte és el que ens permetrà satisfer les necessitats que proposa el pilot. Dissenyar i implementar un sistema de monitoratge autònom que sigui capaç de guardar dades sobre el recorregut de la suspensió, la posició del gas i la inclinació de la moto.

Per complementar aquests objectius generals, se n'han plantejat d'altres amb la finalitat de millorar el rendiment i la funcionalitat del prototip. L'emmagatzematge de les dades en una targeta SD, ens evitarà haver de desmuntar l'aparell cada vegada que se'n vulguin extreure les dades. La programació del software per processar aquestes dades s'ha de desenvolupar amb la finalitat de mostrar gràfiques on s'hi representin les dades de forma clara i eficient.

Un altre punt important a destacar és que el projecte tindrà una part final d'implementació en el món real. Per tant, caldrà tenir en compte que moltes de les coses que s'hagin vist al laboratori, a la realitat poden variar inesperadament. Es suposa que caldrà una re-avaluació general del projecte un cop es munti a la moto corrent en un circuit real.

Tot i que no se'n va fer constància a la presa de decisions inicials, la incorporació d'un mòdul GPS al projecte ens proporcionarà dades molt interessant a l'hora de prendre mesures al món real. La intenció d'afegir aquest element és de guardar la posició geogràfica de la moto en tot moment i poder manipular les dades per poder obtenir la forma del circuit i representar-hi les dades de velocitat.

Per tant, com a resum, podem dir que els objectius més important són:

- Monitoratge de suspensió, gas i inclinació.
- Obtenció de dades GPS
- Sistema d'emmagatzematge extraïble.
- Representació de les dades de forma gràfica.

1.4. Estructura del document

Aquest projecte proposa una solució de baix cost per a un sistema de monitoratge, que conjuntament amb els sensors adequats compleix els objectius marcats.

En el capítol 2, es fa una descripció general del muntatge. Primer es veu el diagrama de blocs per tenir clar els components i la comunicació entre ells; posteriorment es fa una explicació de

cada component de forma general, sense entrar en detalls tècnics.

El capítol 3 explica pas per pas el procés d'implementació. Comença repassant com s'han testejat i integrat els diferents components que conformen el prototip final fins a arribar a la integració total del sistema. A mesura que s'avança en la implementació, es mostren gràfiques amb els resultats obtinguts.

El capítol 4 és dels més complexes del treball. Fa una descripció de la metodologia emprada per calcular la inclinació de la moto. Presenta la solució analítica i els diagrames adequats per facilitar-ne la comprensió.

El disseny final del prototip es descriu al capítol 5. És aquí on es resolen els dubtes sobre on es col·locarà el prototip i quin serà l'encapsulat que el contindrà, entre d'altres.

El capítol 6 explica quina és l'estructura dels mòduls de programació que formen el projecte i quina funció té cadascun d'ells.

Finalment al capítol 7 s'expliquen les diferents conclusions que s'han assolit realitzant el projecte, així com els entrebancs que s'han anat trobant i un breu apartat de treball futur.

Figura 1.2.: Yamaha YZ450Z

2. Descripció del muntatge

2.1. Diagrama de blocs

Per obtenir una primera visió de l'estructura del muntatge final, el diagrama de blocs de la figura 2.1 representa les connexions entre components de forma general.

Les fletxes que connecten els components indiquen en quin sentit s'envia la informació. El bloc central representa el cervell del projecte; és l'encarregat de gestionar tota la informació que envien els diferents sensors i mòduls i transferir les dades a la targeta SD.

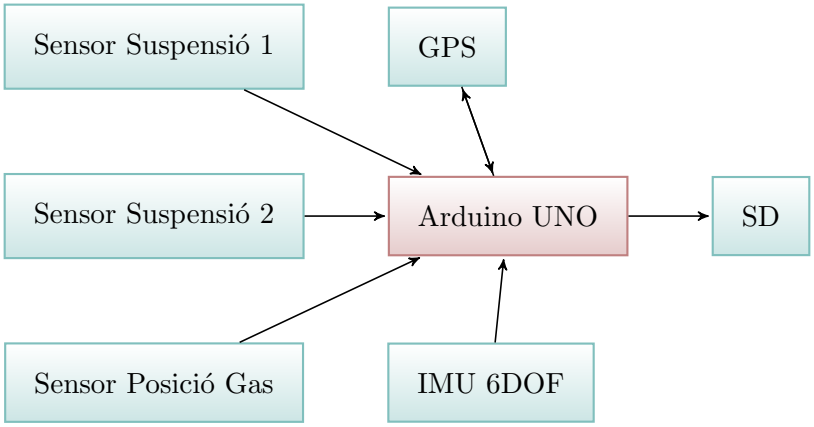


Figura 2.1.: Diagrama de blocs del projecte

2.2. Components

Quan parlem de *hardware* ens referim a totes les parts físiques d'un sistema informàtic i/o electrònic els components del qual són elèctrics, electrònics, electromecànics o mecànics. En el nostre cas, aquests elements ens permeten governar el sistema i adquirir tota la informació necessària per complir els requisits desitjats. A continuació es fa una descripció dels elements principals que componen el *hardware* del nostre sistema.

2.2.1. Arduino UNO

`Arduino` [Ard16] és una plataforma *open source hardware* d'origen italià dissenyada amb l'objectiu de fer més simple i accessible el disseny de circuits electrònics amb microcontroladors. La plataforma busca oferir facilitat alhora de crear nous dispositius amb sensors i actuadors, fet que ha suposat que a l'actualitat la usin des de persones a nivell amateur, fins a una gran quantitat de professionals.

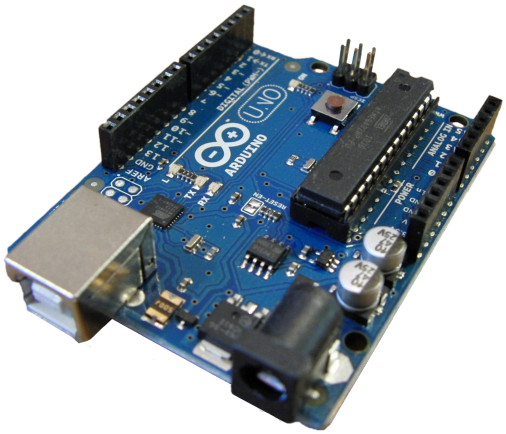


Figura 2.2.: Placa Arduino UNO

Actualment existeixen un gran nombre de models de plaques programables al mercat; cadascuna d'elles amb unes característiques específiques i diferents de la resta. Pel projecte actual es farà servir el model `Arduino UNO` que consisteix en un microcontrolador de la família Atmel AVR ATmega328p. Incorpora un total de 20 pins dels quals 6 són analògics i 14 digitals (I/O)[1]. Aquests estan distribuïts en ports on cada port conté un nombre específic de pins. La placa facilita una interfície USB amb la finalitat de comunicar-se, via sèrie, amb un computador. Aquest mateix bus de comunicació permet la programació del microcontrolador.

Per tal de programar la placa, s'utilitza l'entorn de desenvolupament propi de la plataforma anomenat `Arduino IDE`; un *software open source* el llenguatge de programació del qual es basa en C/C++ i ofereix una sèrie de llibreries per interactuar amb diferents elements exteriors a la placa, com per exemple un mòdul SD, tot i que es poden escriure llibreries pròpies.

Finalment comentar que la placa es pot alimentar de vàries maneres. La opció més usada alhora d'experimentar és el propi cable USB que connecta amb l'ordinador. Per altra banda, quan es requereix de mobilitat, es sol optar per piles alcalines, piles recarregables o bateries LiPo; aquesta opció requereix d'un voltatge d'entrada entre 6 i 18V, connectat al *plug* de connexió per a fonts d'alimentació externes a través d'un adaptador.

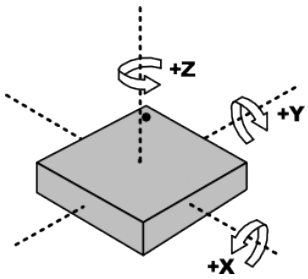
2.2.2. IMU 6DOF

Una Unitat de Mesura Inercial o IMU (de l'anglès *Inertial Measurement Unit*), és un dispositiu electrònic que mesura les velocitats, orientació i forces gravitacionals d'un aparell, usant una combinació d'acceleròmetres, giroscopis i baròmetres.

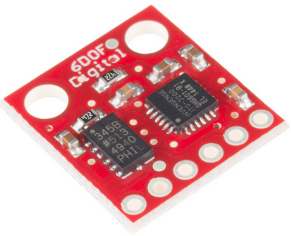
En aquest projecte es treballa amb una IMU de 6 graus de llibertat (figura 2.3b). El model utilitzat és de la marca *Sparkfun* i combina un acceleròmetre ADXL345 [Dev16] i un giroscopi ITG3200 [Inv16]. Cadascun dels tres eixos de cada sensor està disposat de forma ortogonal amb la resta (figura 2.3a). Es comunica amb el microcontrolador a través del protocol I^2C.

[1] I/O = Entrada/Sortida

L'acceleròmetre és un instrument que mesura acceleracions. Ens proporciona la informació de 3 eixos amb una alta resolució (13 bits) i possibilitat de mesurar fins a ±16g. Per altra banda, el giroscopi mesura velocitats angulars. També ofereix informació de 3 eixos mesurant fins a ±2000º/s.



(a) Eixos de coordenades IMU



(b) IMU 6DOF *Sparkfun*

Figura 2.3.: *Inertial Measurement Unit*

Un dels principals inconvenients que presenten aquests mòduls és que normalment són afectats per un error acumulatiu. Degut a que el sistema està agregant els canvis detectats a les posicions prèviament calculades, qualsevol error en la mesura, per petit que sigui, es va acumulant punt a punt.

2.2.3. Sensor de suspensió

El sensor de suspensió [2dd16a] és un potenciòmetre lineal que ens permet calcular el recorregut de la suspensió. S'alimenta a 5 volts i consta d'una vara que es desplaça segons la posició de la tija de la suspensió.

És el component més car de tot el projecte; per això, el Jaume va contactar amb un seu conegut que treballa en una escuderia de Moto2 per parlar sobre la possibilitat de deixar-nos-en un durant el temps de desenvolupament del projecte. Després de varis dies ens van oferir la possibilitat d'utilitzar uns potenciòmetres lineals de 75mm i 155mm de recorregut de la marca *2d-datarecording*, una de les marques de monitoratge més bones del mercat.

Figura 2.4.: Sensor de suspensió *2d-datarecording*

El sensor utilitza un connector Binder 719, SPM de 5 vies; aprofitarem aquest tipus de connexió pel prototip final perquè és segura i deixa el component molt subjecte i evitem haver de fer malbé el cable original.

2.2.4. Sensor de posició del gas

El sensor de posició del gas [Aut16], també conegut com a TPS (*Throttle Position Sensor*), s'encarrega de monitoritzar la posició de la papallona d'entrada d'aire cap al motor. És un simple potenciòmetre rotacional que es desplaça de 0 al voltant dels 120 graus. La majoria de TPS s'alimenten a 5 volts i consten d'una resistència variable que regula la tensió a la sortida segons la posició del gas.

Figura 2.5.: Imatge sensor TPS

2.2.5. Mòdul GPS

El sistema GPS (*Global Positioning System*) [Wik16b] va ser desenvolupat entre els anys 1973 i 1993 pel Departament de Defensa dels Estats Units d'Amèrica. Es basa en una constel·lació de 24 satèl·lits, distribuïts en 6 òrbites al voltant d'uns 26km d'altura respecte la Terra, que permeten calcular la posició geogràfica de qualsevol punt sobre aquesta.

Els satèl·lits incorporen rellotges atòmics amb la finalitat d'obtenir un alt nivell de sincronisme; treballen a una freqüència de 1575.42MHz per a ús civil i a 1227.60MHz per a ús militar.

Les dades que rebem a través del nostre mòdul GPS segueixen el protocol NMEA (*National Marine Electronics Asociation*). Totes les sentències tenen dues lletres que defineixen el tipus de dispositiu que utilitzarà les dades. Per exemple, els receptors comercials tenen assignat el prefix `GP` seguit de tres lletres més que defineixen el contingut de la trama. D'aquesta manera el receptor pot decidir quina sentència vol acceptar.

Cada sentència comença amb el símbol $ i acaba amb un retorn de carro \n. Les dades es separen en comes amb la finalitat de facilitar-ne la lectura i el posterior tractament.

Un cop estudiades les diferents sentències NMEA que existeixen i tenint en compte les dades que ens són necessàries pel projecte, es va optar per treballar exclusivament amb la NMEA GPRMC (*Recommended minimum specific GPS/Transit data*) d'on se n'extreuen les dades de latitud, longitud i velocitat. La sentència presenta una forma com la següent:

```
$GPRMC,192740.500,A,4152.5896,N,00202.2964,E,3.78,339.72,210416,,,A*6B
```

- **RMC**: Recommended Minimum Specific GPS/Transit Data.
- **192740.500**: Hora en format UTC (19h 27m 40.500s)
- **A**: Valida les dades de posició. Una V indicaria posició invàlida.

- **4152.5896, N**: Indica la latitud (41º 52.5896' Nord)
- **00202.2964, E**: Indica la longitud (2º 2.2964' Est)
- **3.78**: Velocitat en nusos
- **339.72**: Rumb en graus respecte la horitzontal
- **210416**: Data en format UTC (21/04/2016)
- **A*6B**: Checksum

Un cop entesa la trama de dades, necessitem proveir-nos d'un mòdul GPS que ens proporcioni aquesta informació. Una primera recerca fa veure que la majoria de mòduls del mercat treballen a una freqüència d'1Hz, insuficient pel cas que ens correspon. Per veure-ho més clar, vegem quina distancia recórrer aproximadament una supermotard en 1 segon, suposant que va a una velocitat de 90km/h:

$$Distancia = Velocitat * Temps = 25[m/s] * 1[s] = 25m$$

Aquesta distància entre mostres és molt significativa al nostre projecte; per tant s'ha buscat una solució i s'ha trobat el mòdul *Adafruit Ultimate GPS* [Ada16] que ens permet augmentar la freqüència de mostreig a 10Hz; per tant, si tornem a fer els càlculs anteriors:

$$Distancia = Velocitat * Temps = 25[m/s] * 0.1[s] = 2.5m$$

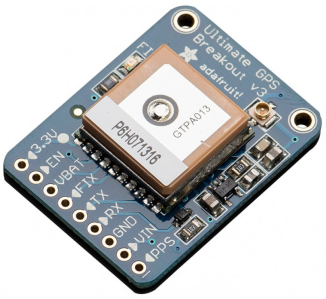


Figura 2.6.: Mòdul GPS *Ultimate Adafruit GPS*

2.2.6. Mòdul SD

Una targeta SD (*Secure Digital*) [Dam16] és una targeta de memòria utilitzada per emmagatzemar continguts en dispositius portàtils. Alhora de seleccionar una targeta, cal tenir en compte una sèrie de conceptes tècnics: la mida, la capacitat d'emmagatzematge i la velocitat a la qual pot copiar i transmetre dades.

Pel que fa la mida, actualment existeixen 3 tipus de targeta: SD, mini SD i micro SD.
Si ens fixem amb la capacitat d'emmagatzematge, en trobem 3 tipus diferents:

- **SD SC *Standard Capacity***: Permet emmagatzemar fins a 2Gb.

- **SD HC *High Capacity***: Fins a 32Gb.
- **SD XC *Extended Capacity***: Capaces d'emmagatzemar fins a 2Tb.

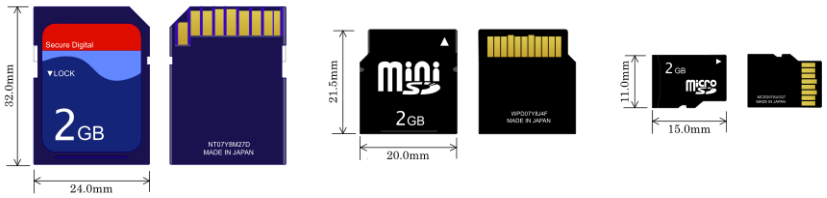


Figura 2.7.: Mides de targetes SD, mini SD i micro SD

L'últim concepte, el de velocitat, es subdivideix en dues parts; la classe i la velocitat del bus. La velocitat mínima amb la qual una targeta SD guarda les dades s'anomena classe. Cal tenir en compte que aquesta velocitat depèn també de la velocitat del dispositiu transmissor. Trobem quatre tipus de classe: 2, 4, 6 i 10 que es relacionen directament amb la velocitat: 2, 4, 6 i 10Mb/s o més en la classe 10. Per tal de transmetre les dades de la targeta a un dispositiu (per exemple l'ordinador) hem de tenir en compte la velocitat del bus. Totes les targetes de classe 6 o inferiors utilitzen el bus estàndard, però a les de classe 10 podem trobar busos d'alta velocitat o ultra alta velocitat UHS-I i UHS-II, que suporten velocitats de 10Mb/s i 30Mb/s respectivament.

3. Implementació

Aquest apartat pretén fer una descripció més detallada dels passos d'implementació que es van seguir amb cadascun dels components, a través d'exemples i aspectes tècnics que ens permetin veure el procés de principi a fi.

El programa principal del dispositiu es composa d'un sol mòdul anomenat `arduino.ino`. Posteriorment es fa un processat de les dades a través d'un conjunt de mòduls `Python` [Fou16], un llenguatge d'alt nivell creat per Guido van Rossum a finals dels anys vuitanta. Aquest llenguatge de programació es caracteritza per una sintaxi molt neta propera al llenguatge natural i un codi fàcilment llegible. És multi plataforma, el que ens permet executar el nostre codi en diferents sistemes operatius sense haver de realitzar pràcticament cap modificació. Tot i que en algun moment de l'explicació es mostren parts del codi, als annexes del treball s'hi troba el codi dels mòduls.

Una manera d'assegurar que els resultats obtinguts són correctes durant la implementació, és enviar les dades a l'ordinador a través del port sèrie i guardar les dades a un arxiu de text i fer una gràfica de les dades rebudes a través d'un petit `script Python`.

3.1. IMU i comunicació

Amb la finalitat de poder connectar la IMU a l'Arduino i extreure'n les dades, el primer pas que es va dur a terme va ser l'estudi del protocol de comunicació I^2C.

3.1.1. Protocol I^2C

El protocol I^2C (*Inter-Integrated Circuit*) [Ele16] és un bus de comunicació molt utilitzat per comunicar circuits integrats; un dels usos més comuns és la comunicació entre un microcontrolador i sensors perifèrics.

A nivell de *hardware* (figura 3.1) la comunicació consta de dues senyals: SCL (Senyal de `clock`) i SDA (Senyal de dades). Per establir connexió amb els esclaus, primer de tot el màster envia una trama d'adreça, indicant amb quin esclau té intenció d'interactuar. Seguidament s'envien les trames de dades, de l'esclau al màster o a l'inrevés seguint unes normes específiques del protocol.

A l'hora d'implementar el protocol fem ús de la llibreria `<Wire.h>` que ens proporciona les funcions necessàries per iniciar el protocol i establir la connexió amb els esclaus. La llibreria s'encarrega de gestionar totes les casuístiques que puguin sorgir. D'aquesta manera, per exemple, no ens hem de preocupar de gestionar els bits de control. Tot i això cal respectar els pins de connexió a l'Arduino; en el cas particular del protocol I^2C la senyal SDA correspon al pin

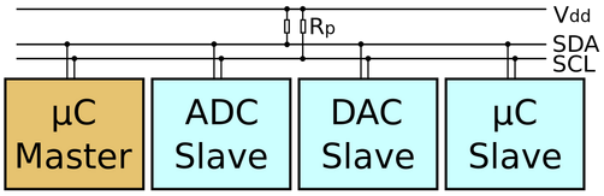


Figura 3.1.: Esquema protocol I^2C

analògic A4 i la senyal SCL al pin analògic A5.

Per optimitzar el codi s'han creat dues noves funcions que simplifiquen la comunicació I^2C permetent enviar i llegir dades de forma més eficient, evitant així la repetició de línies de codi innecessàries. Aquestes funcions ens permeten simplificar la configuració inicial de l'acceleròmetre i el giroscopi entre d'altres.

La inicialització del protocol es realitza a la funció `setup` del programa, establint una velocitat de comunicació de 400kHz. Aquesta velocitat és característica del mode *fast speed* del protocol i és la màxima suportada per l'Arduino, tot i que existeix el mode *high speed* que permet velocitats entre 1.7 i 3.4MHz.

Cal tenir present que cada vegada que s'accedeix a una interrupció es bloquegen la resta automàticament. El protocol I^2C treballa per interrupcions i és cridat dins la interrupció del TimerOne, per tant caldrà activar interrupcions tan bon punt entrem a la funció de *callback* del TimerOne per permetre captar les dades de la IMU. Aquesta pràctica pot ser una xic perillosa en segons quins àmbits, però en la nostra aplicació en concret no suposa un perill extremadament elevat i menys després de les proves que s'han realitzat.

Aquestes interrupcions es generen cada 8ms, el que equival una freqüència de 125Hz com es mostra a la figura 3.2.

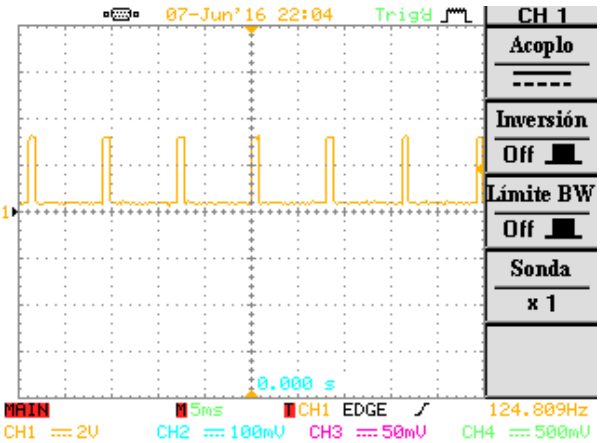


Figura 3.2.: Temps entre interrupcions a l'oscil·loscopi

3.1.2. IMU 6DOF

Un cop explicat el protocol de comunicació entre l'Arduino i la IMU, és hora de la configuració. Des de la mateixa funció `setup` on inizialitzem l'I^2C, també es criden les funcions d'inicialització de l'acceleròmetre [5He16a] i el giroscopi [5He16b]. Aquestes funcions ens permeten configurar els registres dels sensors per obtenir-ne un funcionament adequat. Per realitzar aquesta fase, l'Arduino inicia la comunicació, indica sobre quin registre del sensor vol escriure i li envia el byte adequat.

Tant l'accelerò metre com el giroscopi tenen moltes opcions de configuració, però les més importants i útils pel projecte són:

Acceleròmetre

- **0x2D** `POWER_CTL`: Controla l'alimentació. Escrivint el valor 0x08, activem el sensor i evitem que entri en modes de baix consum o inclús s'apagui.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Figura 3.3.: Registre 0x2D Accelerò metre

- **0x31** `DATA_FORMAT`: Controla el format de les dades que es troben als registres 0x32 a 0x37. Escrivint el valor 0x08, obtindrem dades de resolució màxima entre ±2g.

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Figura 3.4.: Registre 0x31 Accelerò metre

Giroscopi

- **0x3E** `POWER_MANAGEMENT`: Controla l'alimentació del sensor. Escrivint el valor 0x00, forcem una configuració idèntica a l'accelerò metre. A través dels 3 bits de menor pes podem especificar sobre quin *clock* treballa el dispositiu: intern, amb referència a un dels eixos o extern.

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
3E	62	H_RESET	SLEEP	STBY_XG	STBY_YG	STBY_ZG	CLK_SEL			00h

Figura 3.5.: Registre 0x3E Giroscopi

- **0x16** `DLPF`: Controla el format de les dades que es troben als registres 0x1D a 0x22. Escrivint el valor 0x19, obtenim un rang de treball de ±2000º/s indicat als bits 4 i 5 i una F_i de 1kHz combinat amb un filtre passa baix a 188Hz segons els tres bits de menor pes.

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
16	22	-			FS_SEL		DLPF_CFG			00h

Figura 3.6.: Registre 0x16 Giroscopi

- **0x15** `SAMPLE_RATE_DIVIDER`: Determina la freqüència de mostreig del dispositiu a partir de la fórmula $F_s = F_i/(SAMPLE\_RATE\_DIVIDER{+}1)$ on F_i s'obté de la configuració del registre 0x16. Si escrivim el valor 0x07, obtenim una freqüència de mostreig de 125Hz:

$$F_s = \frac{1kHz}{(7+1)} = 125Hz$$

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
15	21	SMPLRT_DIV								00h

Figura 3.7.: Registre 0x15 Giroscopi

En aquesta primera fase d'experimentació, les dades són rebudes en paquets de 8 bits, o sigui 1 byte; un cop llegides, es guarden a una variable del tipus `int16_t` fent un desplaçament a l'esquerra del byte de major pes i concatenant el de menor pes fent una `or` amb els 8 bits de menor pes de la variable. Posteriorment s'envien les dades pel port sèrie separades per comes per facilitar-ne la interpretació.

Mantenint el sensor el més pla possible, els resultats obtinguts van ser els mostrats a les figures 3.8 i 3.9. A simple vista es pot apreciar que el giroscopi es veu més afectat per soroll.

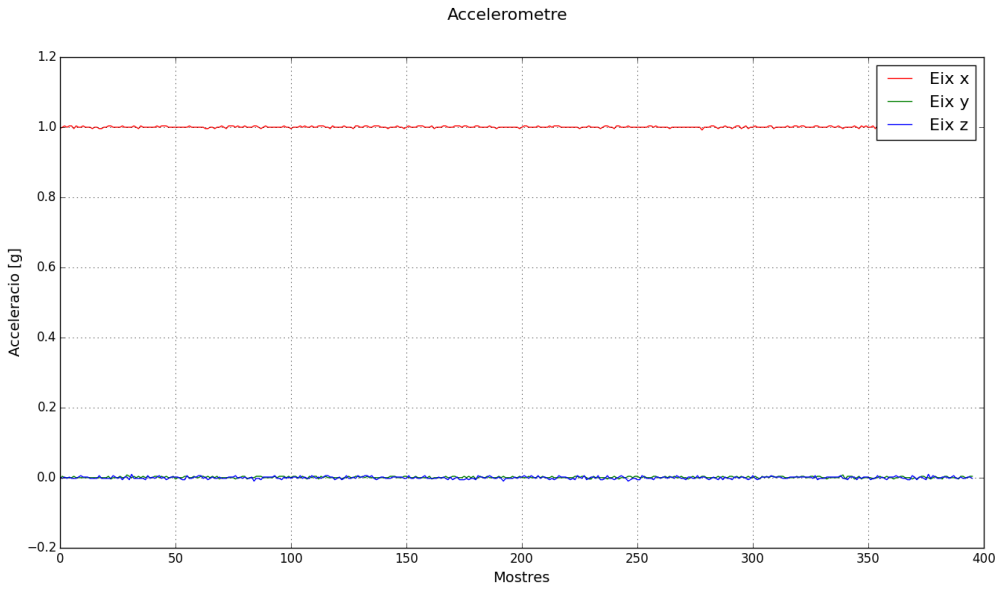


Figura 3.8.: Gràfica acceleròmetre

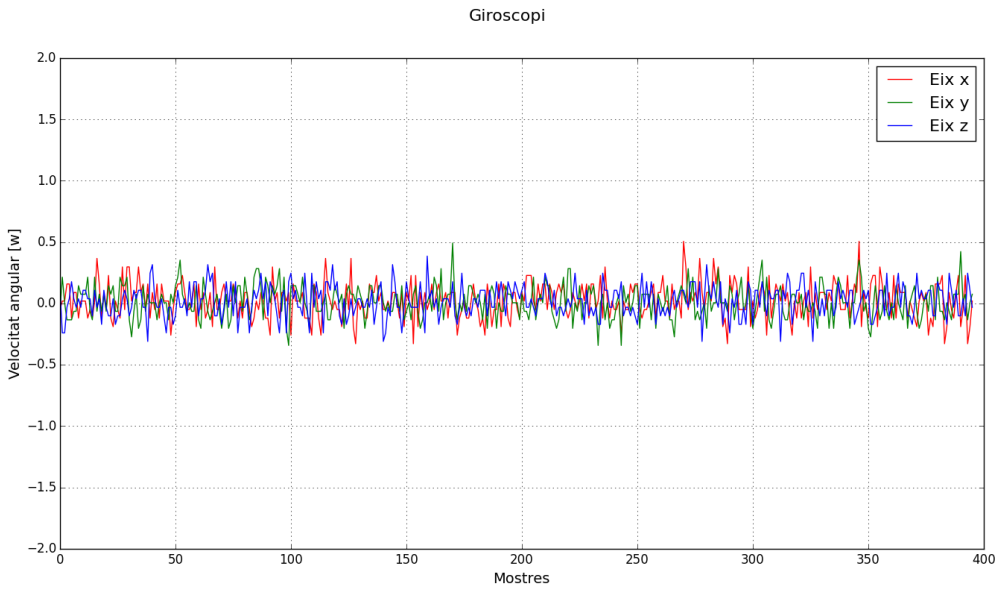


Figura 3.9.: Gràfica giroscopi

Si repetim el procés, però aquesta vegada movem el sensor aleatòriament, el resultat obtingut és el mostrat a les figures 3.10 i 3.11. En aquest segon cas no s'aprecia el soroll del giroscopi observat a la gràfica anterior perquè l'escala de valors de l'eix Y ha canviat de [-2, 2] a [-600, 800] perquè les gràfiques estan escalades automàticament, però cal tenir present n'hi continuem tenint en tot moment.

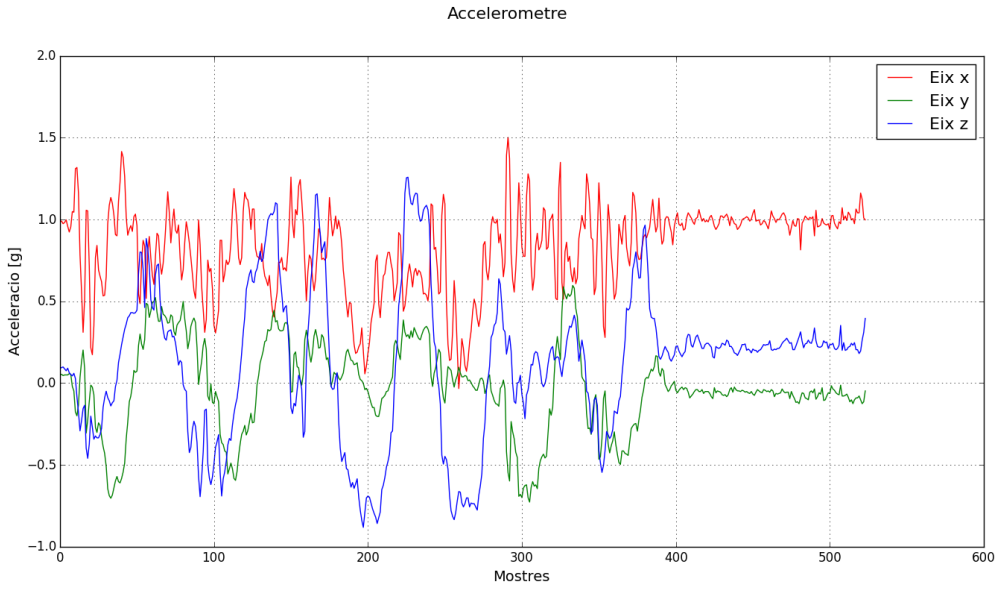


Figura 3.10.: Gràfica acceleròmetre dades aleatòries

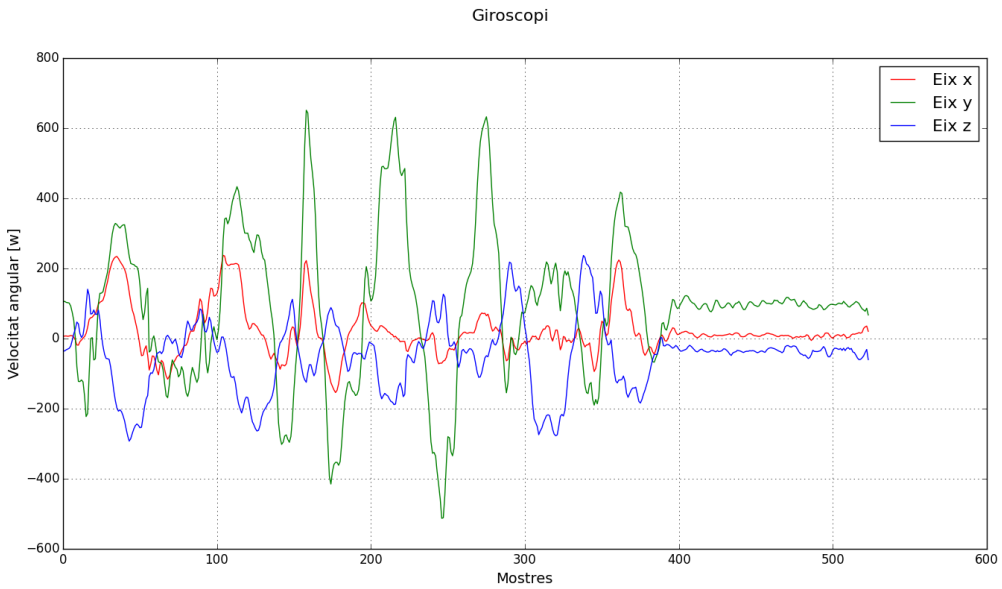


Figura 3.11.: Gràfica giroscopi dades aleatòries

L'extracció de l'angle d'inclinació d'aquest conjunt de dades, és un procés suficientment extens com per dedicar-hi un apartat exclusiu del treball (capítol 4).

Les dades dels eixos del sensor estan separades en dos bytes[1]. La combinació d'aquests dos bytes per formar una dada del tipus int16_t per obtenir el valor mesurat, s'especifica al *datasheet*[2] de cada component. Així doncs el primer byte de cada eix de l'acceleròmetre és el de menor pes i la conversió es duria a terme fent:

```
Xa = (int)byte(1) << 8 | byte(0)
```

Mentre que en el giroscopi, el primer byte de cada eix es el major pes i per tant la conversió queda:

```
Xg = (int)byte(0) << 8 | byte(1)
```

El que es duu a terme és aquesta combinació dels dos bytes formant una dada del tipus int16_t. Per tal d'alliberar l'Arduino de fer aquests càlculs, emmagatzemarem cada byte per separat i els processarem amb Python. Aquest tipus de dades pot prendre valors entre -32678 i +32678, mentre que sense signe pren valors de 0 a 65535. Com que Python no suporta els tipus de dades int16_t, sino que treballa amb dades sense signe (uint16_t), el que sha fet és realitzar la combinació i comprovar si el resultat és major que 32678; en cas que es compleixi la condició, es resta 65535 per obtenir el valor amb signe:

```
Xa = int(byte(1)) << 8 | int(byte(0))
  if (Xa > 32678):
    Xa -= 65536
```

[1] 1 byte = 8 bits
[2] Document que especifica el funcionament i altres característiques d'un component.

Per posar un exemple, el valor binari `1111111110100100` codificat amb signe té un valor de -92, mentre que codificat sense signe té un valor de 65444; per tant veient que es compleix la condició, es fa la operació $65444 - 65536$ i s'obté el valor -92.

3.2. Sensor de suspensió

Un cop es van aconseguir els sensors de suspensió, el primer que es va fer va ser comprovar-ne l'estat. Per tal de verificar que funcionaven correctament, es van mesurar les resistències entre borns per deduir-ne l'esquema elèctric del potenciòmetre. Els resultats obtinguts van ser els mostrats a les taules 3.1 i 3.2. Com a la majoria de potenciòmetres, es va suposar que l'esquema elèctric constava d'una resistència variable i una petita resistència al born de sortida. Recordem que el sensor utilitza un connector Binder 719 de 5 pins, dels quals només n'utilitza tres distribuïts de la següent manera:

- **Pin 1**: GND
- **Pin 2**: V_{cc}
- **Pin 5**: V_{out}

Observant les dades vam veure que el sensor de 75mm no funciona correctament. Tot i això, com que la moto porta suspensions de motocròs, aquest sensor no ens serveix donat el poc recorregut que té; però si que vam poder aprofitar el sensor de 155mm per la suspensió de darrera. Pel que fa la suspensió de davant, es va prescindir de monitoritzar-la tot i que s'ha deixat el projecte apunt per si s'incorpora el sensor en un futur.

Terminals	Entrat	Sortit
1 - 2	3.23kΩ	3.26kΩ
1 - 5	1.14kΩ	4.82kΩ
2 - 5	1.46kΩ	3.94kΩ

Taula 3.1.: Mesures Sensor Suspensió 75mm

Terminals	Entrat	Sortit
1 - 2	5.96kΩ	5.96kΩ
1 - 5	6.57kΩ	0.68kΩ
2 - 5	0.61kΩ	6.63kΩ

Taula 3.2.: Mesures Sensor Suspensió 155mm

Les dades són llegides aprofitant la interrupció de la IMU a través d'un dels pins analògics de l'Arduino i convertides a centímetres posteriorment aplicant la funció 3.1. El valor de la dada que llegim es converteix d'analògic a digital a través de l'ADC que incorpora l'Arduino. Aquest procés es realitza de la mateixa manera pel sensor de posició del gas que s'explica al següent apartat.

Un ADC (*Analog-to-Digital Conversor*) [Álv16] és un dispositiu electrònic capaç de convertir una senyal analògica, en una senyal digital. Aquesta conversió consta en realitzar, de forma periòdica, mesures de l'amplitud (tensió) d'una senyal analògica d'entrada. Seguidament es fa un procés de quantificació, que consisteix en convertir valors continus en valors discrets seleccionant per aproximació dins el marge de resolució que ofereix l'ADC. Un cop realitzada la quantificació, només falta la codificació, que és la traducció dels valors obtinguts al procés anterior a codi binari. Tot el procés queda esquematitzat a la figura 3.12.

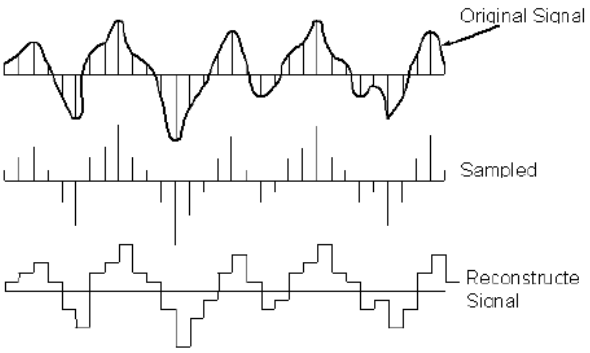


Figura 3.12.: Digitalització d'una senyal analògica

L'Arduino UNO incorpora un ADC de 10 bits. Per poder transformar aquests 10 bits en un byte i emmagatzemar la dada a la cua, un cop hem obtingut el valor de l'ADC a través de la funció `analogRead(pin)`, desplacem els bits dues posicions a la dreta i posteriorment ho guardem a la cua, perdent els bits número 9 i 10, però mantenint els de major pes.

```
sps = analogRead(A1) >> 2;
queue.enqueue(sps);
```

Seguidament es representen les dades en una gràfica (figura 3.13). Aplicant la conversió següent:

$$Recorregut[cm] = L_{max} - \frac{ADC \cdot 5}{255} \cdot \frac{L_{max} - L_{min}}{5} - L_{min} \tag{3.1}$$

On L_{max} és la llargada màxima de la tija del sensor (18.8cm), L_{min} és la llargada mínima de la tija del sensor (2.7cm), `ADC` és el valor llegit per l'Arduino, `5/255` és la conversió a volts i `5` és el voltatge màxim que podem llegir del sensor.

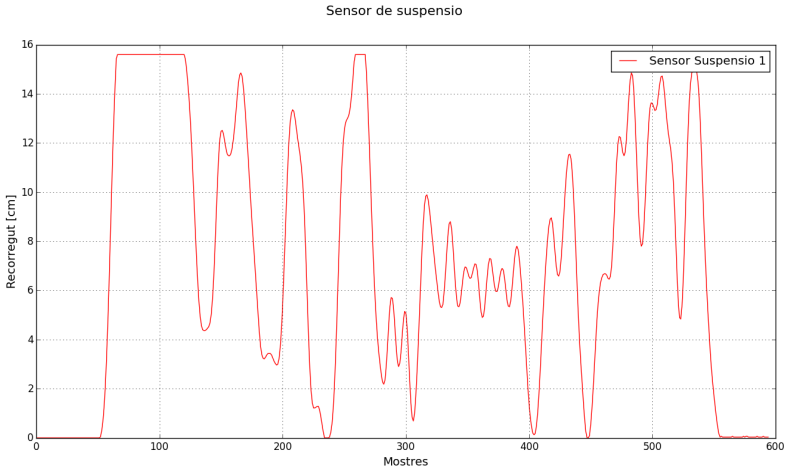


Figura 3.13.: Gràfica testeig Sensor Suspensió

3.3. Sensor de posició del gas

Aquest element és present en totes les motos d'injecció on es controla l'obertura de l'aire al motor a través de la centraleta. Per tal de no desmuntar el sensor que ja incorpora la moto, per fer les proves es va fer servir un TPS de les mateixes característiques que va aconseguir el Jaume.

Seguint el procediment descrit amb el sensor de suspensió, s'han comprovat les resistències entre borns i s'han plasmat a la taula 3.3. Aquesta vegada el connector consta de 3 pins distribuïts de la següent manera:

- **Pin 1** (Negre): GND
- **Pin 2** (Blau): V_{cc}
- **Pin 3** (Groc): V_{out}

Terminals	Tancat	Obert
1 - 2	4.52kΩ	4.52kΩ
1 - 5	0.18kΩ	4.67.kΩ
2 - 5	4.54kΩ	0.18kΩ

Taula 3.3.: Mesures Sensor Posició del Gas

Observem la bona funcionalitat del TPS, per tant, es procedeix a incorporar-lo al sistema i extreure'n dades.

Seguint el procés descrit en l'apartat del sensor de suspensió, les dades es llegeixen aprofitant la interrupció de la IMU i convertides d'analògic a digital a través de l'ADC que incorpora l'Arduino.

En aquest cas, quan mesurem 0v significa que no hi ha gens de gas donat, per tant l'obertura que hem de veure és de 0º; en canvi, quan mesurem 3.20v ens indica que el gas està obert del tot, el que equivaldria a una rotació aproximada de 125º. Per determinar els graus d'obertura del sensor a partir de la lectura del pin analògic, s'aplica la fórmula 3.2.

$$Obertura[^{o}] = \frac{ADC \cdot 5}{255} \cdot \frac{Gmax}{3.3} \tag{3.2}$$

On `ADC` es el valor llegit per l'Arduino, `5/255` és la conversió a volts, `Gmax` son els graus d'obertura màxima (125º) i `3.3` és el voltatge màxim que podem llegir del sensor. La lectura d'aquest sensor es fa idènticament seguint el procediment descrit a l'apartat anterior.

Tot seguit es representen les dades a la gràfica 3.14, fent joc amb la rotació. En aquest cas, hem de fixar l'escala de la gràfica entre amb uns valors a l'eix Y de [0, 140] per veure el resultat adequat.

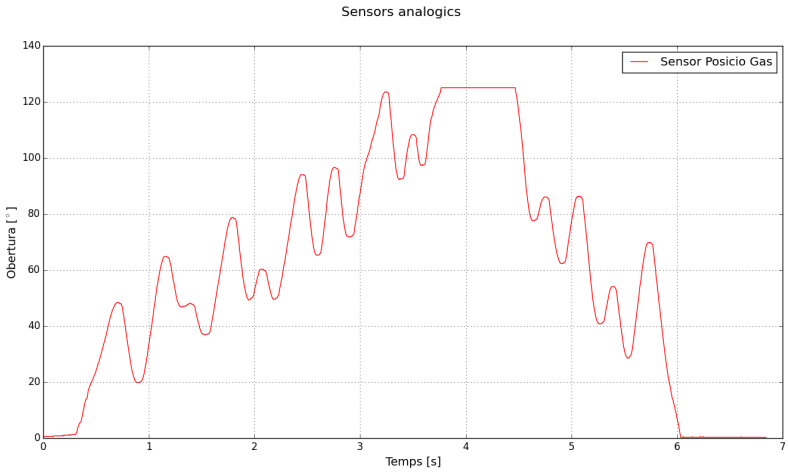


Figura 3.14.: Gràfica testeig TPS

En el resultat final, la gràfica dels sensors de suspensió i posició del gas es representaran conjuntament en dos eixos Y amb escales diferents (figura 3.15). En el nostre cas, cal recordar que un dels dos sensors de suspensió no funcionarà i el connectarem a 0.

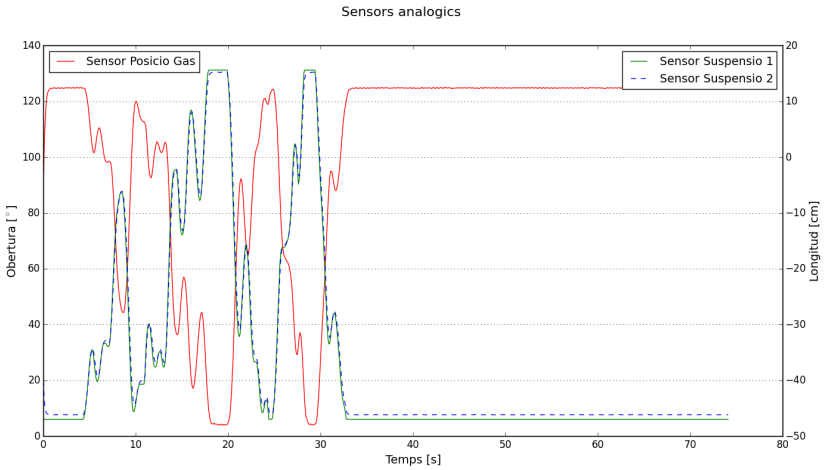


Figura 3.15.: Gràfica sensors analògics

3.4. SD i comunicació

Per tal de poder comunicar l'Arduino amb el el mòdul SD i enviar-hi les dades, el primer pas que es va dur a terme va ser l'estudi del protocol de comunicació SPI.

3.4.1. Protocol SPI

El bus SPI (*Serial Peripherical Interface*) [Pro16] és un protocol de comunicacions síncron entre dispositius electrònics a molt curta distància. Pot assolir velocitats molt elevades comu-

nicant un microcontrolador amb varis esclaus en mode *full dúplex*[3].

A nivell de *hardware* (figura 3.16) la comunicació consta de quatre senyals: CLK (Senyal de `clock`) que correspon al pin 13 de l'Arduino, MOSI (*Master Out Slave In*), pin 11, és línia de dades on el màster envia informació a l'esclau, MISO (*Master In Slave Out*), pin 12, és la línia de dades on l'esclau envia informació al màster i SS (*Slave Select*), pin 4, que indica per a quin esclau és el missatge enviat.

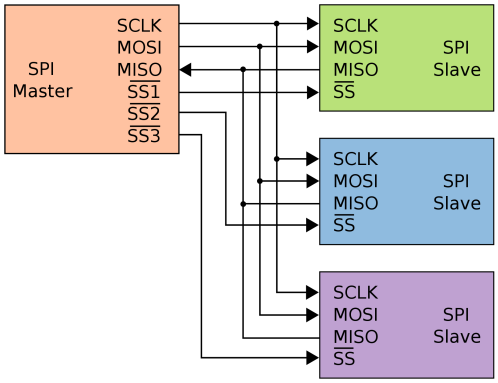


Figura 3.16.: Esquema protocol SPI

Alhora d'implementar el protocol fem ús de la llibreria `<SPI.h>` que ens proporciona les funcions necessàries per iniciar el protocol i seleccionar l'esclau. De la mateixa manera que la llibreria emprada pel protocol I^2C, s'encarrega de gestionar totes les casuístiques que puguin sorgir.

A diferència del protocol I^2C, en aquest cas no cal interactuar directament sobre la llibreria perquè ho farem a traves de la llibreria `<SD.h>`.

3.4.2. Mòdul SD

Per tal de dotar el sistema de memòria pròpia, es va incorporar un mòdul SD [Enr16] la funció del qual és emmagatzemar les dades que llegim dels diferents sensors i el GPS en un fitxer de text separat per comes. Per configurar-lo adequadament es va fer ús de la llibreria `<SD.h>` que ens permet la configuració del protocol de comunicació SPI i la creació i gestió dels fitxers. El nom d'aquest fitxers segueix el format 8.3, que significa que admet fins a vuit caràcters al nom i tres a l'extensió; per exemple `dadesSensors.txt` no seria un nom correcte donat que té 12 caràcters.

Les primeres proves es van realitzar fent la lectura d'un LDR[4] i guardant el valor a un fitxer de la targeta SD. Un cop verificat el funcionament es va passar a combinar-ho amb la IMU 6DOF. Aprofitant el mateix `sketch` que configurava l'AVR per interactuar amb la IMU, s'afegeix el codi necessari per la configuració del mòdul SD; la principal diferència és

[3] A grans trets *full dúplex* significa que pot enviar i rebre dades al mateix temps

[4] Dispositiu electrònic semiconductor amb la capacitat de variar el seu valor de resistència segons la llum incident.

que en lloc d'enviar les dades per port sèrie i veure-les a la pantalla de l'ordinador, s'envien a l'arxiu de la targeta SD. Tot i que el temps d'obertura i tancament del fitxer és costós (uns 17ms), és necessari fer-ho cada vegada que hi emmagatzemem dades, perquè no controlem el moment el qual el dispositiu deixarà de funcionar i, si no tanquem el fitxer, les dades es perden.

Un exemple de fitxer enregistrat a la targeta microSD amb les dades de la IMU tindria un aspecte com el següent, on cada línia consta de sis valors corresponents als eixos de l'acceleròmetre i el giroscopi respectivament.

```
248,0,-24,-44,119,84
252,0,-27,-36,104,115
242,0,-23,-16,-17,205
233,2,-30,-23,-283,444
241,1,-33,-84,-608,710
265,-18,-35,-197,-1333,910
264,-25,-31,-400,-2497,1193
332,-58,-39,-409,-3109,1117
334,-64,-68,-265,-3115,722
294,-47,-92,-199,-3050,556
197,-4,-121,-119,-3127,657
140,21,-105,-58,-3706,1075
```

El primer problema sorgeix quan es calculen els temps de treball del mòdul SD. Guardar les dades al fitxer té un cost de temps relativament petit (uns 2.5ms), mentre que obrir i tancar l'arxiu per guardar-hi les dades, com s'ha comentat abans, té un cost d'entre 15 i 20ms. Sabent això, es podria pensar que amb obrir el fitxer una sola vegada n'hi hauria prou, però cal recordar que si no es tanca el fitxer i la targeta queda perd l'alimentació, les dades es perden.

Per solucionar aquest problema, s'ha optat per treballar amb cues. Pel que fa al GPS, l'Arduino gestiona un *buffer* propi per guardar les dades rebudes pel port sèrie; el problema que va sorgir és que té una mida de 64 bytes, insuficient per guardar trames senceres de GPS que són d'uns 72 bytes aproximadament. Per tant, s'ha modificat la llibreria que gestiona el *hardware* de l'Arudino augmentant aquest *buffer* a 256 bytes.

Per altra banda, les dades que llegim del sensor d'inclinació i els sensors analògics cada vegada que salta una interrupció, són guardades a una cua FIFO[5] (figura 3.17) que es gestiona a través de la llibreria `QueueArray.h` [Cha16].

Després de fer les primeres proves, es va descobrir un comportament inusual de la llibreria, la qual canviava la mida de les cues a mesura que s'omplien i es buidaven. Aquest fet podia comportar sobrepassar la capacitat de la memòria de l'Arduino fent que tot el sistema deixés de funcionar. La modificació de la llibreria evitant aquest comportament en va solucionar el problema.

[5]En català "Primer a entrar, primer a sortir", és un concepte usat en estructura de dades i teoria de cues. Guarda analogia amb les persones que esperen en una cua i van sent ateses en l'ordre en què van arribar, és a dir, que la primera persona que entra és la primera persona que surt

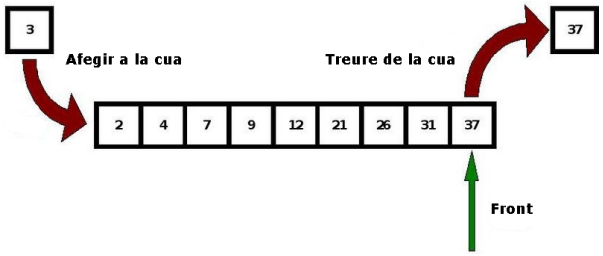


Figura 3.17.: Esquema d'una cua FIFO

Per altra banda, el fet de guardar totes les dades en un sol fitxer podia crear problemes d'accés si, per exemple, el *buffer* del sèrie s'havia de buidar i a l'hora també ho volia fer la cua. Per tant es va decidir treballar amb dos fitxers, un pel GPS anomenat `gps.txt` i un segon pels sensors analògics i la IMU anomenat `sensors.txt`.

Per entendre el funcionament de tot plegat, hem de tenir present que el *buffer* del port sèrie i la cua dels sensors van creixent a mesura que rebem dades i són buidades al loop del programa quan l'Arduino té temps de fer-ho. No es pot determinar amb exactitud cada quan es gravaran dades a la SD perquè dependrà de si la cua és plena o buida en el moment de qüestionar si té dades disponibles i si en aquell moment en té més o menys per guardar.

Per veure tot aquest procés de sincronisme es va fer ús de l'oscil·loscopi i així controlar els temps de treball de cada component. A la figura 3.18 es pot veure la interrupció que recull dades de la IMU i els sensors analògics de color groc, el buidatge del *buffer* del port serie i l'accés i guardada de dades al fitxer `gps.txt` de color rosa i finalment el buidatge de la cua i l'accés i guardada de dades al fitxer `sensors.txt` de color blau.

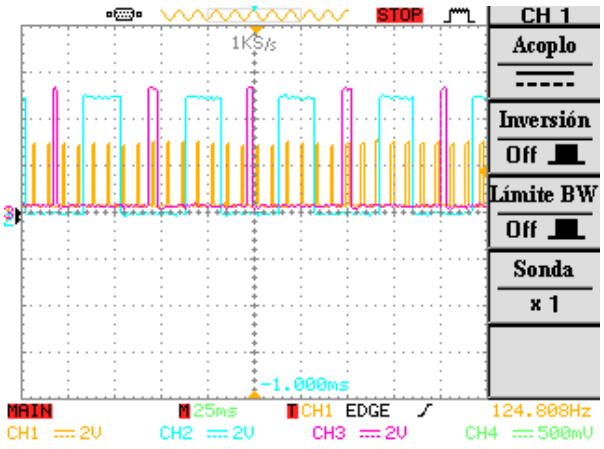


Figura 3.18.: Oscil·loscopi

3.5. GPS i comunicació

3.5.1. Comunicació Sèrie

Un port[6] sèrie [Lla16] envia informació mitjançant una seqüència de bits. Es necessiten almenys dos connectors per per realitzar la comunicació de dades: RX (recepció) i TX (Transmissió). Moltes vegades es referencien els port sèrie com UART (*Universally Asynchronous Receiver/Transmitter*), una unitat que incorporen alguns processadors, encarregada de realitzar la conversió de les dades a una seqüència de bits y transmetre'ls o rebre'ls a una velocitat determinada.

Per altra banda existeix el terme TTL (*Transistor-Transistor Logic*). Això significa que la comunicació es realitza a través de variacions de senyal entre 0V i V_{cc} (on V_{cc} sol ser 3.3V o 5V).

En el nostre cas, l'Arduino UNO incorpora una UART que opera a nivell TTL (0V-5V) on els pins de transmissió i recepció corresponen als pins 0 (RX) i 1 (TX). Per iniciar el protocol, només fa falta determinar la velocitat de comunicació:

```
void setup() {
  Serial.begin(9600)
  Serial.flush() //Netegem el port
}
```

3.5.2. Mòdul GPS

La millor manera per començar a testejar el mòdul GPS és connectar-lo a l'ordinador a través del convertidor sèrie - USB que incorpora l'Arduino. Seguint les connexions de la figura 3.19, l'Arduino no intervé directament en la comunicació; simplement fa la funció de convertidor.

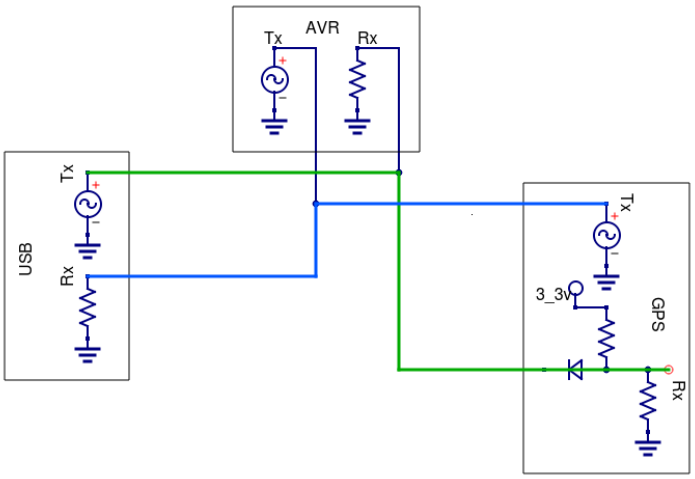


Figura 3.19.: Esquema de connexions GPS: Primer cas

[6]Nom genèric amb el qual denominem les interfícies, físiques o virtuals, que permeten la comunicació entre dos ordinadors o dispositius.

El mòdul envia les dades a través d'interrupcions generades per un `clock` pròpi. Per defecte, les envia per port sèrie a 9600 bauds a una freqüència d'1Hz.

Per veure les dades per pantalla, carreguem un `sketch` buit a l'Arduino, obrim un terminal i llegim el port sèrie amb el programa `Picocom`. El resultat és el següent:

```
$picocom -b 9600 -r -l /dev/cu.usbmodem1411

$GPGGA,235945.799,,,,,0,00,,,M,,M,,*73
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,235945.799,V,,,,,0.00,0.00,050180,,,N*4A
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
(*)
$GPGGA,203937.000,4152.6216,N,00202.2910,E,1,06,1.32,551.5,M,51.2,M,,*6F
$GPGSA,A,3,01,08,14,22,11,10,,,,,,,1.63,1.32,0.94*06
$GPRMC,203937.000,A,4152.6216,N,00202.2910,E,0.05,135.34,010616,,,A*6C
$GPVTG,135.34,T,,M,0.05,N,0.09,K,A*31
```

Observem que el GPS envia per defecte el conjunt de sentències NMEA més usades, les quals no mostren cap mena d'informació vàlida fins que s'han trobat satèl·lits (*) i s'ha calculat la posició geogràfica. En bones condicions (cel clar, a l'exterior i sense interferències), el mòdul triga uns 45 segons a determinar la posició on es troba.

L'objectiu d'aquest apartat és configurar el GPS perquè enviï la sentència NMEA GPRMC el més ràpid possible, a una freqüència de 10Hz. Llegint les especificacions del mòdul, s'observa que a la màxima freqüència només pot enviar un tipus de sentència, fet que no ens suposa cap problema pel nostre projecte. Per configurar el mòdul GPS, s'envien un altre tipus de sentències anomenades PMTK. En el nostre cas, farem ús de les següents sentències:

- PMTK API SET NMEA OUTPUT: Indica al GPS que només ha de processar la sentència GPRMC, això ho fem enviant la trama "$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29 \r\n"pel port sèrie.
- PMTK SET NMEA UPDATERATE: Estableix la freqüència d'enviament de les sentències NMEA. En el nostre cas, enviant la trama "$PMTK220,100*2F\r\n", aconseguirem una frqüència de 10Hz.

És molt important l'ordre d'enviament de les sentències PMTK. Es va comprovar al laboratori, que configurar la velocitat d'enviament abans de configurar el tipus de sentència NMEA no té conseqüències al GPS perquè és incapaç de processar més d'una NMEA a una freqüència de 10Hz.

Modificant les connexions com es mostra a la figura 3.20 i connectant el mòdul SD a l'Arduino, podem guardar les dades a mesura que són enviades.

En aquest cas, l'USB no intervé per res en la comunicació, només l'utilitzem per programar l'AVR. Cal tenir en compte que la connexió Rx (GPS) - Tx (AVR) durant la càrrega de l'`sketch` de configuració de la plataforma n'impedeix el procés fent saltar l'error `avrdude:`

`stk500_getsync() attempt 1 of 10: not in sync: resp = 0x00`. Per evitar aquesta situació, la millor solució és apagar completament el GPS connectant el pin `EN` a `GND` i procedir a la càrrega.

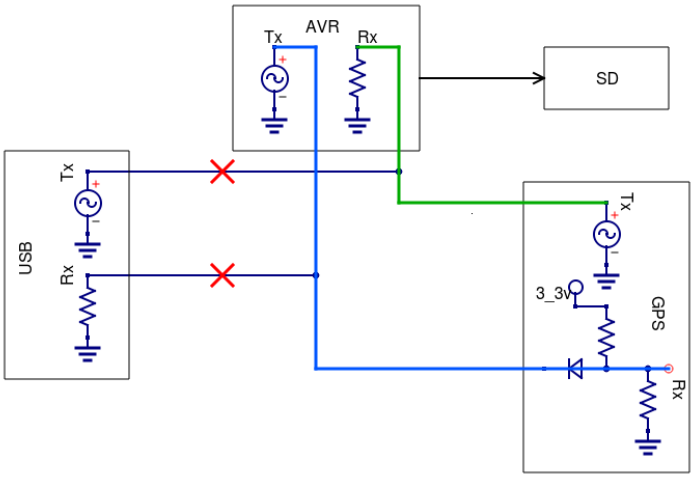


Figura 3.20.: Esquema de connexions GPS: Segon cas

Per veure més clar el resultat de les dades que ens dóna el GPS, es va fer un recorregut de prova guardant les dades a la targeta SD i convertint-les a un fitxer `.kmz`[7] amb l'aplicació web `gpsvisualizer.com`. El resultat sobre Google Earth (figura 3.21) ens permet observar la imprecisió del modul a falta d'una antena externa, tot i que vam quedar gratament satisfets del resultat obtingut.

Figura 3.21.: Recorregut de prova plasmat sobre Google Earth

Com que pel projecte no és necessari mostrar les dades del GPS sobre un mapa físic, si no obtenir la forma del circuit, aquest imprecisió no suposa un greu problema.

[7]Fitxer del programa Google Earth

Per representar les dades del GPS i obtenir la traçada del circuit, plasmarem les coordenades de latitud i longitud sobre un pla cartesià convertint les dades a graus decimals:

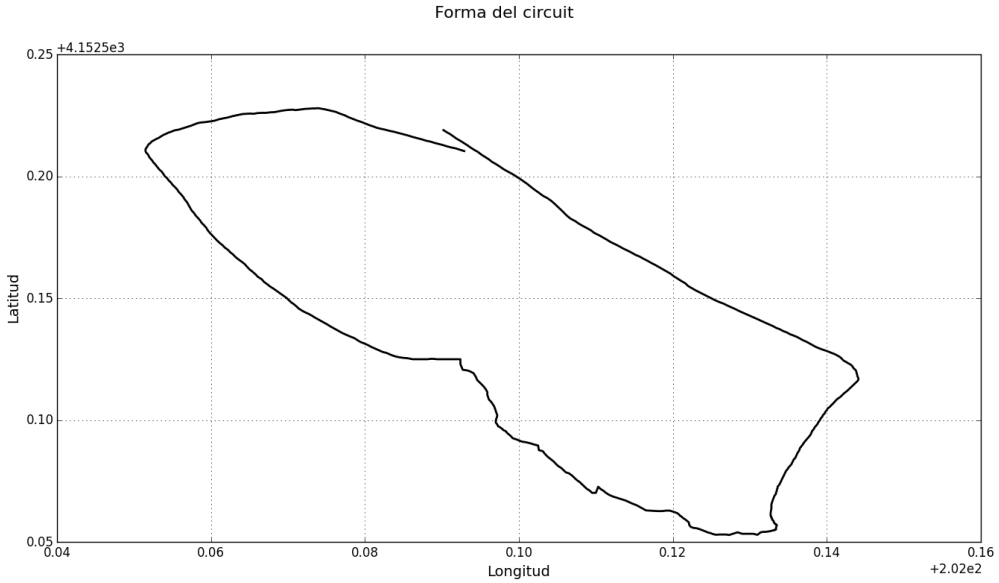


Figura 3.22.: Traçada del circuit

Per aprofitar encara més les dades que rebem del mòdul GPS, es representa la velocitat sobre la mateixa traçada a partir d'una escala de color (figura 3.23). Cal recordar que les unitats de les dades de velocitat que rebem del mòdul són en nusos i per tant, caldrà aplicar la conversió a km/h sabent que $1(nus) = 1,852(km/h)$.

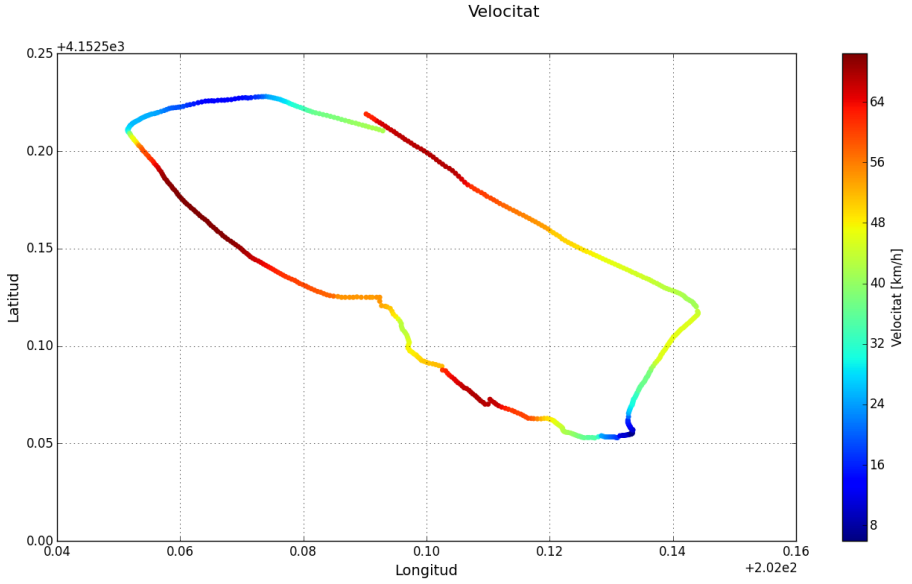


Figura 3.23.: Traçada del circuit segons velocitat

Per representar les dades segons la distància recorreguda al circuit (figura 3.24), s'aplicarà la Fórmula de Haversine [Wik16a] (3.3), una fórmula molt important per a la navegació que ens permet calcular la distància entre dos punts del globus terrestre coneixent-ne la latitud i longitud.

$$haversin\left(\frac{d}{R}\right) = haversin(\varphi_1 - \varphi_2) + cos(\varphi_1) \cdot cos(\varphi_2) \cdot haversin(\Delta\lambda) \tag{3.3}$$

On haversin és la funció de Haversine: $haversin(\theta) = sen^2(\theta/2) = \frac{1-cos(\theta)}{2}$
d és la distància entre dos punts [km]
R és el radi de l'esfera terrestre (6372,8 km)
φ_1 és la latitud del punt 1 [rad]
φ_2 és la latitud del punt 2 [rad]
$\Delta\lambda$ és la diferència de longituds [rad]

Si resolem a partir de la funció arcsinus:

$$d = R \cdot haversin^{-1}(h) = 2 \cdot R \cdot arcsin\left(\sqrt{h}\right) \tag{3.4}$$

on $h = haversin\,(d/R)$

Finalment, desenvolupant ens queda:

$$d = 2 \cdot R \cdot arcsin\left(\sqrt{sin^2\left(\frac{\varphi_1 - \varphi_2}{2}\right) + cos(\varphi_1) \cdot cos(\varphi_2) \cdot sin^2\left(\frac{\Lambda_2 - \Lambda_1}{2}\right)}\right) \tag{3.5}$$

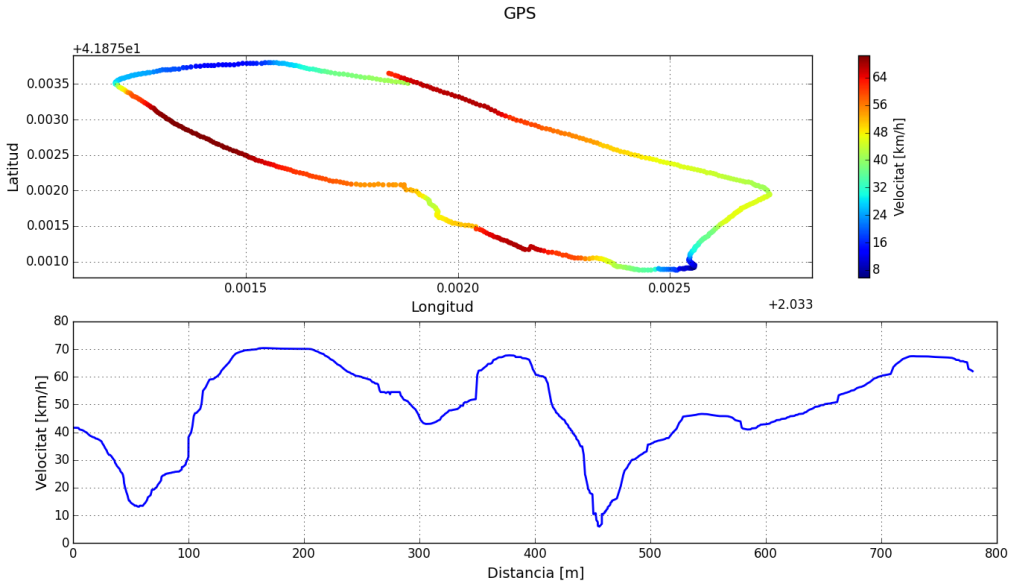


Figura 3.24.: Representació de la velocitat segons distància

4. Metodologia pel càlcul de la inclinació

En un primer moment, quan es va plantejar el problema, va sorgir ràpidament la idea de calcular la inclinació amb un acceleròmetre [Wen16]. Tal i com s'ha explicat anteriorment a l'apartat 3.1, l'acceleròmetre s'usa per calcular acceleracions. La relació entre l'angle d'inclinació i l'acceleració mesurada pel sensor és la gravetat.

La gravetat de la Terra és una acceleració constant que sempre apunta al centre d'aquesta. Quan l'acceleròmetre es troba paral·lel a la gravetat, un dels eixos mesurarà 1g mentre que els altres dos, que es trobaran perpendiculars a la gravetat, mesuraran 0g. Per tant, utilitzant un sol eix de l'acceleròmetre (figura 4.1), l'angle d'inclinació pot calcular-se de la següent manera:

$$\alpha = \sin^{-1}(\text{Acceleracio mesurada}/\text{Acceleracio de la gravetat}) \tag{4.1}$$

El problema ve donat quan es calculen acceleracions superiors a 1g, molt freqüents quan l'objecte es troba en moviment; llavors quan volem calcular l'angle, la funció $\sin^{-1}$ no es pot calcular perquè $\sin(\alpha) = x$, on $-1 < x < 1$.

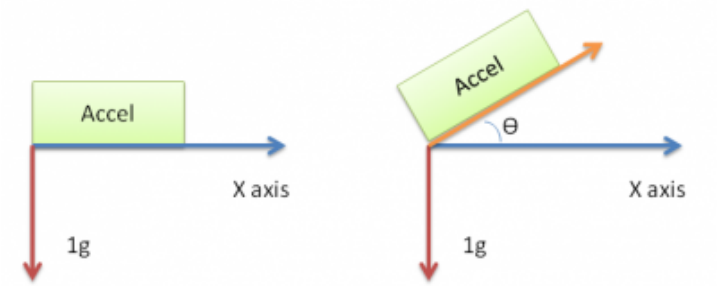


Figura 4.1.: Càlcul de l'angle d'inclinació amb un eix

Una manera més precisa de calcular l'angle d'inclinació és a través de dos dels eixos de l'acceleròmetre (figura 4.2). Les diferències entre les dues equacions es poden veure a la gràfica 4.3. Per implementar la segona equació, es va fer ús de la funció `atan2` que ens permet passar com a paràmetres les dades dels eixos per separat i ens retorna el quadrant on es troba la solució.

$$\alpha = arctan\left(\frac{x}{z}\right) \tag{4.2}$$

Desafortunadament, aquesta teoria tan sols pot aplicar-se quan l'objecte està estàtic. Quan l'objecte es troba en moviment, ja hem vist que poden sorgir valors més grans que 1g i a més a més, hi poden haver altres forces d'acceleració actuant sobre ell i interferint en el càlcul de la inclinació, com per exemple l'acceleració centrípeta.

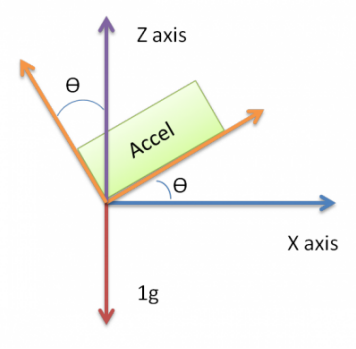


Figura 4.2.: Càlcul de l'angle d'inclinació a partir de dos eixos

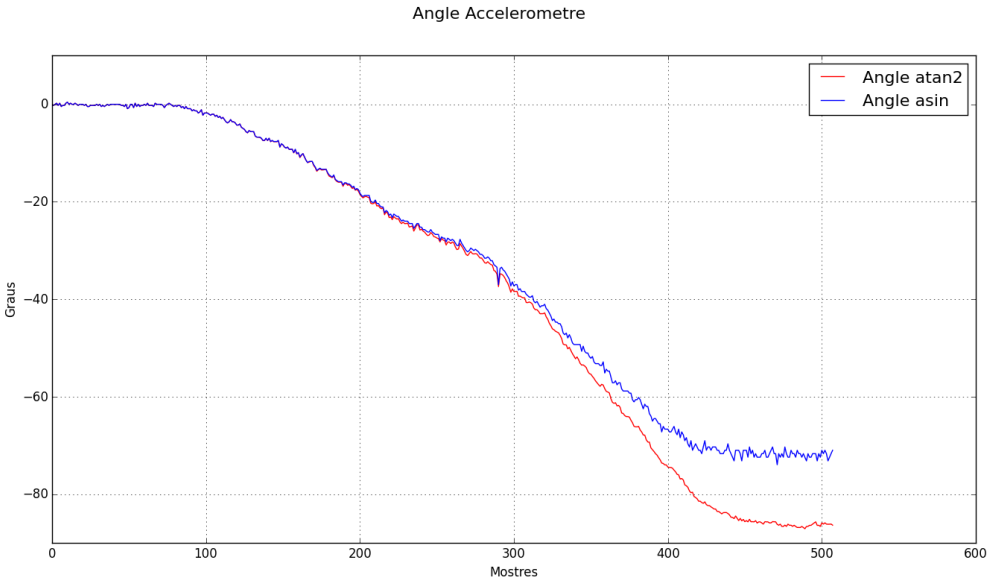


Figura 4.3.: Comparació entre l'equació 4.1 i l'equació 4.2

Per evitar aquesta interferència degut a l'acceleració centrípeta, es va calcular l'angle d'inclinació a partir de dos eixos plantejant el sistema d'equacions següents:

$$\left.\begin{aligned} A_x &= g \cdot cos(\alpha) + A_c \cdot sin(\alpha) \\ A_z &= g \cdot sin(\alpha) + A_c \cdot cos(\alpha) \end{aligned}\right\} \tag{4.3}$$

Desenvolupant el sistema s'obté l'equació 4.4, una equació no lineal que després d'intentar resoldre-la amb l'ajuda de `Matlab` introduint-li varis valors, s'arriba a la conclusió que quan actua acceleració centrípeta sobre el mòdul la solució és doble; en canvi, quan no hi actua acceleració centrípeta i es compleix la condició $|A_z| + |A_y| \leq g$ pot no tenir solució.

$$g = A_x \cdot cos(\alpha) - A_z \cdot sin(\alpha) \tag{4.4}$$

Per tant, es va descartar la fórmula i es va optar per rectificar els valors de l'acceleròmetre utilitzant altres mètodes.

Per corregir el soroll present a les dades de l'acceleròmetre, s'hi aplica un filtre passa baix. Típicament, les components d'acceleració provinents a causa del moviment dinàmic apareixen en un període de temps curt, mentre que la gravetat hi és sempre present. Davant aquesta problemàtica, es va decidir implementar un filtre passa baix IIR de primer ordre sobre les dades del sensor que efectua la mitjana de la mostra actual d'entrada i de la mostra de sortida anterior, com il·lustra la figura 4.4 seguint la fórmula següent:

$$y[n] = (1 - \alpha) \cdot x[n] + \alpha \cdot y[n - 1] \tag{4.5}$$

On `y[n]` representa el valor de l'acceleració filtrat, `x[n]` la mostra actual en brut i `y[n-1]` la mostra filtrada en l'instant anterior.

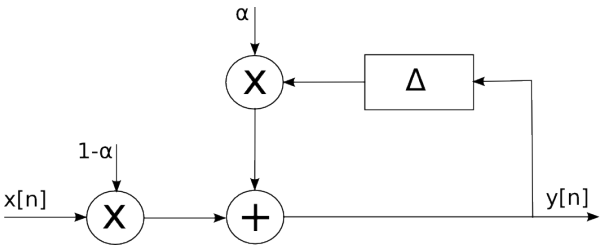


Figura 4.4.: Diagrama de blocs d'un filtre passa baix

El valor α determinarà la freqüència de tall. Com més elevat sigui α més baixa serà la freqüència de tall, per tant, podem filtrar els components d'acceleració d'alta freqüència no desitjats i que ens quedi només l'acceleració deguda a la gravetat. Desafortunadament, l'aplicació del filtre suposa un augment de la latència i el temps de resposta, com es mostra al gràfic 4.5. És aquí on entra en joc el giroscopi.

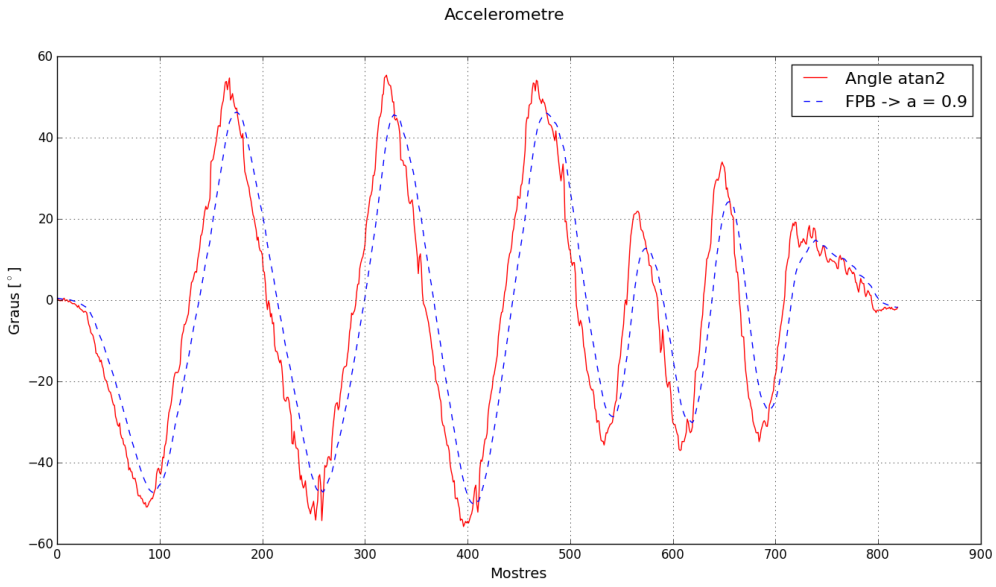


Figura 4.5.: Aplicació d'un filtre passa baix

El giroscopi ens permet calcular velocitats angulars (ω). Per tal d'obtenir l'angle d'inclinació, és necessari integrar les dades provinents del giroscopi seguint l'equació 4.6. El problema en aquest cas és que quan s'integra el valor del giroscopi també s'integra el soroll de la senyal. A més a més, el giroscopi té la limitació que la seva sortida no és constant quan està estàtic. De fet, aquest valor pot canviar inclús pel factor de la temperatura. Aquesta condició és diu deriva i tot i que és molt petita, quan es tracte d'integració, fins i tot el més petit desplaçament farà que les dades integrades tendeixin a l'infinit.

$$\alpha = \int_0^T W_x \, dx \tag{4.6}$$

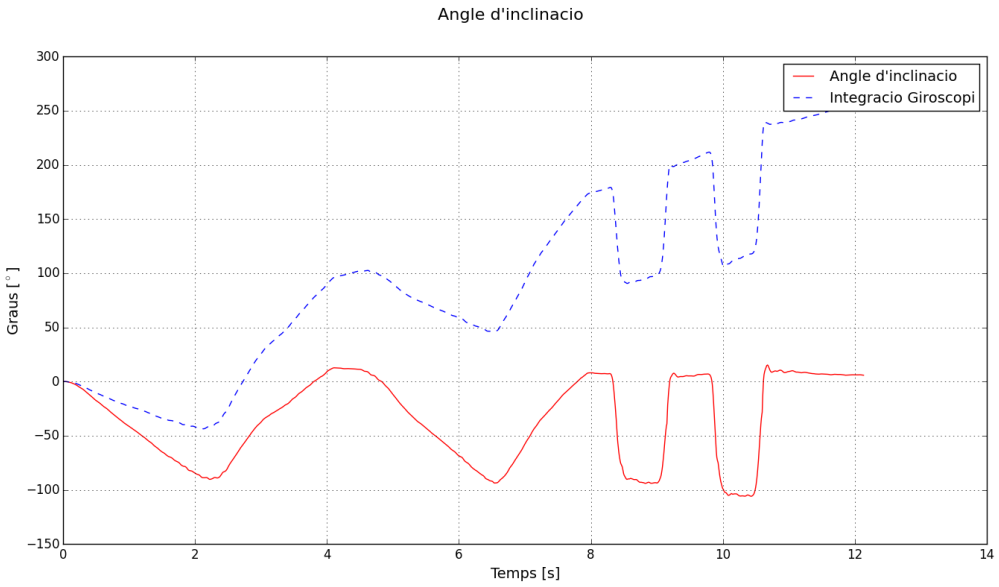


Figura 4.6.: Integració del giroscopi

Després d'estudiar les característiques de cada sensor i veure'n els pros i contres, es procedeix a combinar-los perquè treballin conjuntament. L'acceleròmetre té un temps de resposta lent, mentre que la integració del giroscopi té tendència a desviar-se al llarg del temps. En altres paraules, podríem dir que l'acceleròmetre és útil a llarg termini, mentre que el giroscopi és útil a curt termini.

Una de les maneres de combinar les dades dels dos sensors és a través d'un filtre complementari. Aquest tipus de filtre està dissenyat de tal manera que la fortalesa d'un sensor és utilitzada per combatre la debilitat de l'altre. Aquest tipus de filtre és molt utilitzat en els sistemes de navegació inercial. A causa de la baixa complexitat matemàtica d'aplicació la CPU consumeix pocs recursos computacionals.

En aquest cas, la tasca del filtre complementari és la d'utilitzar la integració del giroscopi (equació 4.6) a curt termini, i després les dades de l'acceleròmetre (equació 4.2) per corregir la desviació a llarg termini. Per implementar el filtre complementari partirem de la següent

fórmula:

$$\alpha = 0.98 \cdot (\alpha_{i-1} + \alpha_{giro} \cdot dt) + 0.02 \cdot \alpha_{acc} \tag{4.7}$$

El filtre complementari aplica un filtre passa baix sobre l'acceleròmetre i un filtre passa alt sobre les dades integrades del giroscopi com esquematitza la figura 4.7.

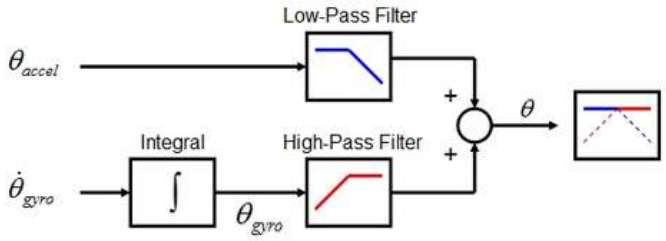


Figura 4.7.: Esquema del filtre complementari

Un cop implementat tot el procés anteriorment descrit, podem veure'n el resultat a la figura 4.8 on s'hi representen els graus d'inclinació calculats per l'acceleròmetre i el giroscopi per separat amb els corresponents filtres ja aplicats i la seva unió mitjançant l'equació 4.7 que descriu el filtre complementari.

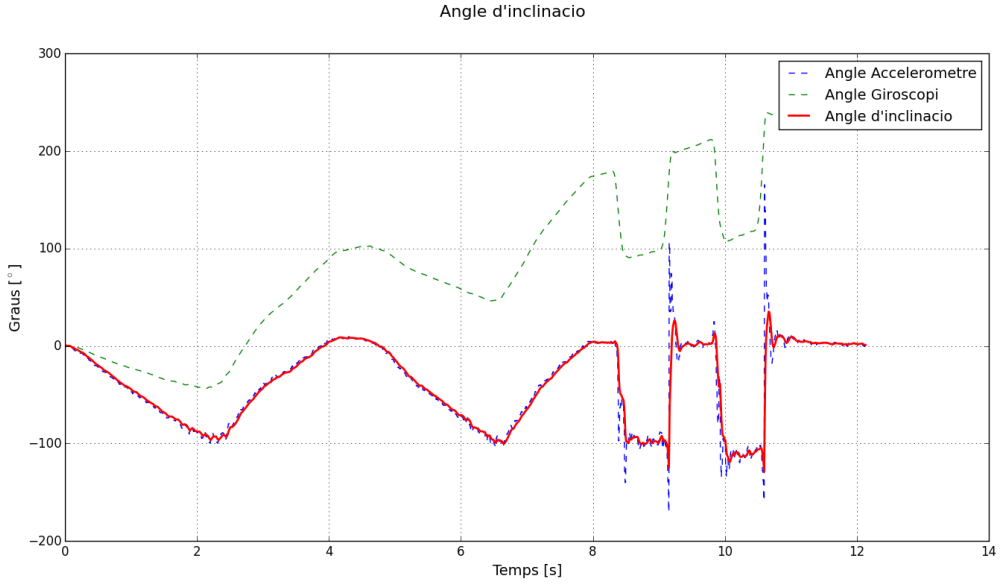


Figura 4.8.: Resultat del càlcul de l'angle d'inclinació

Per tal d'interpretar les dades, cal deixar constància que un angle negatiu indica que la moto s'inclina cap a l'esquerra, mentre que un angle positiu, indica que s'inclina cap a la dreta.

5. Prototip Final

Amb la intenció de crear un producte físic, s'ha dissenyat i construït un prototip d'acer inoxidable on hi col·locarem els components que formen el projecte (apartat 2.2). Es tracta d'un encapsulat on s'hi collarà l'Arduino i s'hi enganxarà la *protoboard*[1] que contindrà els sensors i mòduls. Tindrà quatre sortides a l'exterior: tres pels sensors analògics i una per interactuar amb la targeta SD i evitar haver de desmuntar-ho tot per accedir-hi cada vegada que se'n vulguin extreure les dades.

El primer pas que es va dur a terme va ser fer el disseny 3D del prototip (figura 5.1). Un cop es va tenir el disseny final clar, es va procedir a la construcció. Mitjançant uns petits alçadors de niló, s'evita que les soldadures de l'Arduino facin contacte les unes amb les altres quan toquin la planxa i es curtcircuiti la placa (figura 5.2). Els orificis situats al lateral ens permeten muntar les femelles per la posterior connexió dels sensors analògics.

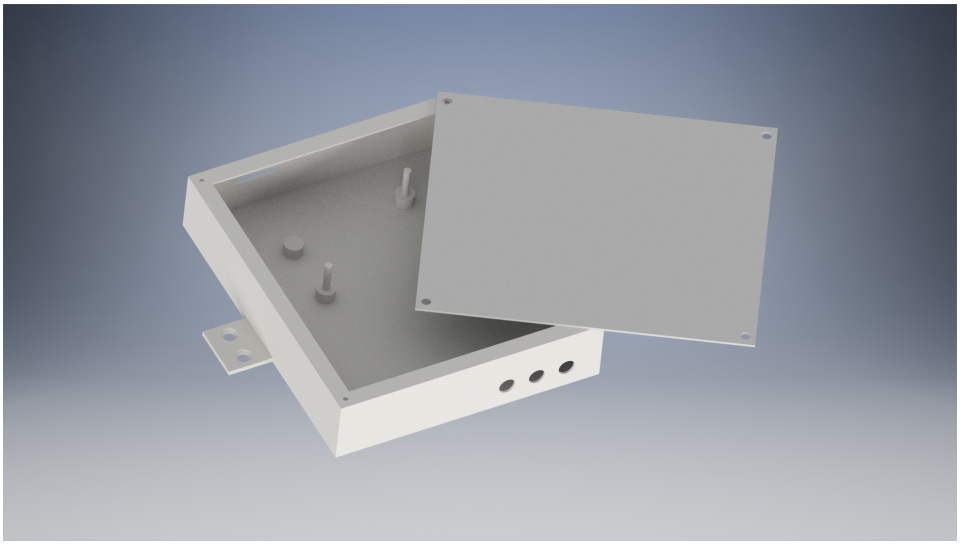


Figura 5.1.: Disseny 3D del prototip

Finalment es munten totes les part sobre l'encapsulat, deixant un prototip totalment funcional apunt pel seu ús.

Un dels aspectes important a tenir en compte és la col·locació del dispositiu. Per tal d'evitar problemes d'escalfament, impactes forts, entrada d'aigua, etc. les grans marques aconsellen ubicar-lo a la part de darrera o bé sota la part frontal del seient del pilot.

[1] En català placa de proves, és una placa d'ús genèric usada per construir prototips de circuits electrònics amb soldadura o sense.

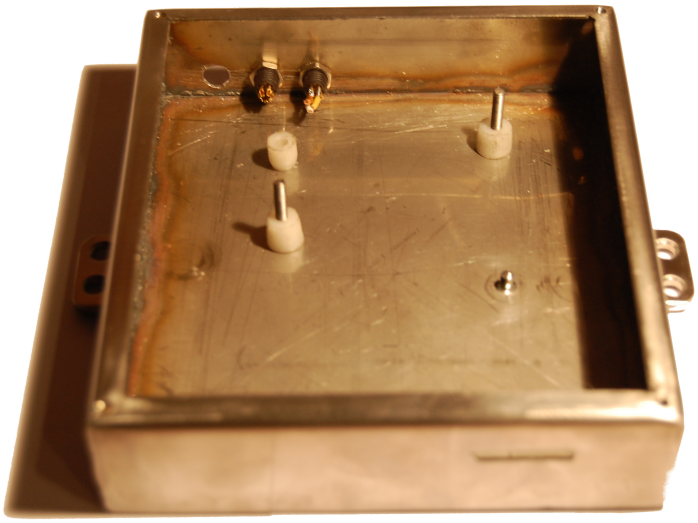


Figura 5.2.: Prototip final sense components

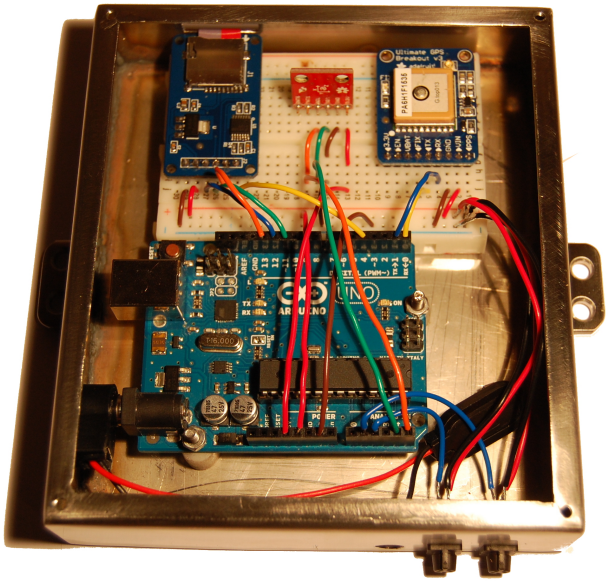


Figura 5.3.: Prototip final

6. Estructura del software

En aquest apartat[1] es descriu el software[2] utilitzat per tal de poder configurar el dispositiu i tractar-ne les dades. La part de programació del dispositiu s'ha programat amb l'IDE d'Arduino, mentre que el processat de les dades s'ha implementat amb `Python`. L'estructua de mòduls per la configuració de l'Arduino es mostra a la figura 6.1 i l'estructura de mòduls que s'ha utilitzat per fer el processament de les dades es mostra a la figura 6.2.

El codi font dels diferents mòduls es troba adjuntat a l'apartat d'annexes II. No s'adjunta el codi de les llibreries perquè són de fàcil accés a la web oficial de l'Arduino i tampoc les he escrit jo mateix. Per tant només s'adjunta el codi propi.

Arduino

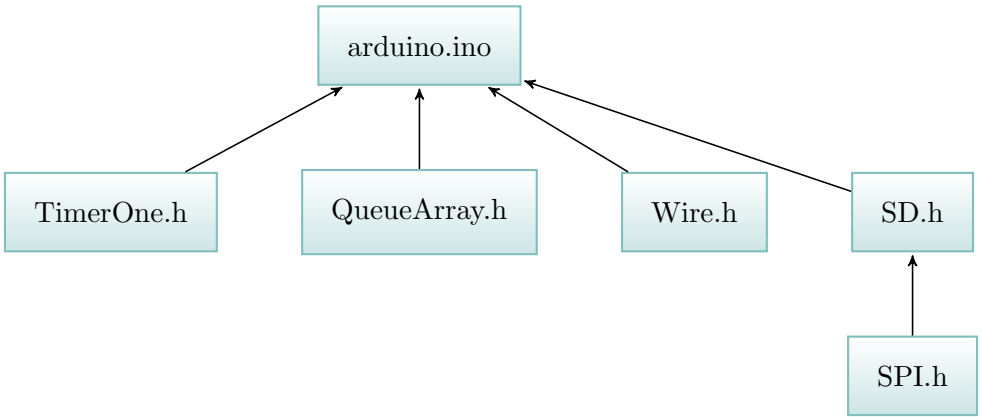


Figura 6.1.: Diagrama de mòduls per configurar l'Arduino

arduino.ino [A] `Sketch` que ens permet programar l'AVR. Fa una lectura dels sensors i en guarda els valors en fitxers segons la font d'on provinguin. Per tal de gestionar tota la informació que rep, fa ús de llibreries tant per la configuració i comunicació amb els sensors, com per la gestió de les dades.

TimerOne.h Llibreria que ens permet configurar les interrupcions a través del Timer1 establint un temps entre interrupcions en microsegons. Per tal d'iniciar la llibreria, es declaren dues variables del tipus `unsigned long` que el timer usarà com a comptador; també es programa la funció de *callback*.

[1] Durant aquest capítol alguns noms de mòduls i alguns fragments de codi, no s'han accentuat segons les regles ortogràfiques, ja que els noms dels mòduls i el codi no poden contenir lletres accentuades.

[2] Segons l'estàndard 729 del IEEE (*Institute of Electrical and Electronics Engineers*) és el conjunt dels programes de computació, procediments, regles, documentació i dades associades que formen part de les operacions d'un sistema de còmput.

QueueArray.h Llibreria que crea i gestiona la cua FIFO on s'emmagatzemaran les dades de la IMU i els sensors analògics mentre aquestes no són gravades a la targeta SD. Aquest tipus de cues tenen unes funcions bàsiques que han de complir per garantir un mínim de funcionament, tals com:

- **queue.init()**: inicialitza la cua posant tots els apuntadors a 0.
- **queue.enqueue()**: afegeix un element al final de la cua.
- **queue.dequeue()**: elimina l'element frontal de la cua, és a dir, el primer que s'ha entrat.
- **queue.front()**: retorna el valor del primer element de la cua.
- **queue.isFull()**: retorna `true` si la cua és plena.
- **queue.isEmpty()**: retorna `true` si la cua és buida.

Wire.h Per tal de configurar l'acceleròmetre i el giroscopi, cal establir una comunicació I^2C; aquesta llibreria ens proporciona les eines per fer-ho amb més facilitat.

SD.h Llibreria que ens gestiona la configuració i accés a la targeta SD, així com la gravada de dades als fitxers. ens permet mantenir un control dels fitxers que conté la targeta i poder eliminar-los, llegir-los i afegir-ne de nous.

SPI.h La llibreria SD.h fa un del protocol SPI per comunicar-se amb el mòdul físic. Aquesta llibraria ens proporciona les eines per establir aquesta connexió.

Python

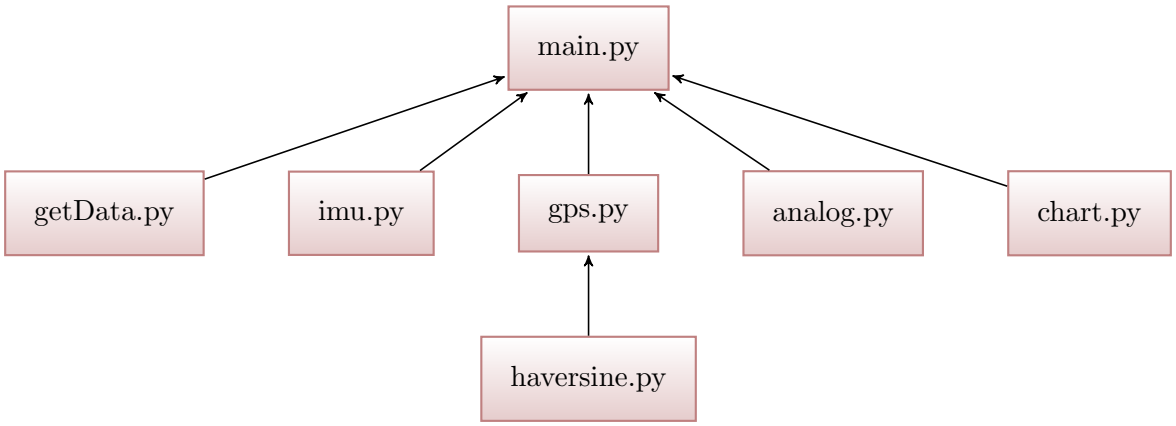


Figura 6.2.: Diagrama de mòduls del projecte

getData.py [B] Mòdul que transforma els fitxers de dades en un diccionari de llistes amb les dades dels diferents sensors per separat: IMU, GPS i sensors analògics. Com a paràmetres rep els dos fitxers passats directament per terminal. El del GPS el llegeix línia a línia, descartant aquelles línies que contenen sentències de configuració PMTK. Per altra banda, el fitxer de la IMU i els sensors analògics el llegeix tot de cop i posteriorment guarda les dades tenint en compte la posició amb la qual estan ubicades dins la trama (figura 6.3) havent calculat el valor correcte a partir dels dos bytes d'informació (en el cas de la IMU).

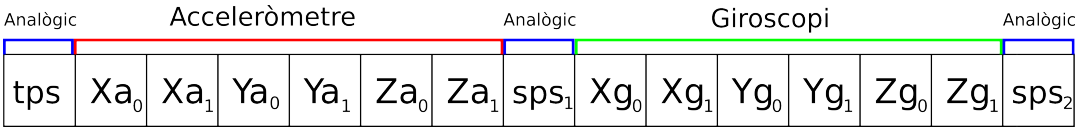


Figura 6.3.: Trama de dades

imu.py [C] És el mòdul encarregat de tractar les dades rebudes de la IMU; calcula l'*offset* fent la mitjana de les deu primeres mostres, en resta el valor corresponent a cada eix i converteix les dades d'acceleració a g's aplicant un factor de conversió de 256 i les dades del giroscopi de velocitat angular a graus/s amb un factor de 14.375. Finalment en calcula l'angle d'inclinació segons l'acceleròmetre, l'angle d'inclinació segons el giroscopi i en combina els resultats a través d'un filtre complementari. Aquest mòdul retorna un diccionari que consta d'una llista amb les dades de l'angle d'inclinació i una llista amb les dades de l'eix de temps.

gps.py [D] Mòdul que tracta les dades rebudes del GPS. Converteix la longitud i la latitud a graus decimals, dades que passa com a paràmetres al mòdul `haversine.py`, i la velocitat a km/h. Retorna un diccionari amb les llistes de les diferents dades.

haversine.py [E] Aquest mòdul permet el càlcul de la distància entre dos punts (latitud i longitud) d'una esfera, en aquest cas la Terra, aplicant la solució analítica descrita a l'equació 3.3.

analog.py [F] És el mòdul que tracta les dades rebudes dels sensors analògics. Transforma els valors de voltatge en graus [º] i centímetres [cm] segons el tipus de sensor. Retorna un diccionari amb les llistes de les dades tractades.

chart.py [G] Donades les dades ja tractades de la resta de mòduls, aquest mòdul en fa les gràfiques. Una primera per l'angle d'inclinació de la IMU, on s'hi representa l'angle d'inclinació; la segona per la velocitat i forma del circuit del GPS i l'última on s'hi representen els tres sensors analògics separats en dues escales de valors, a la dreta els sensors de suspensió i a l'esquerra el sensor de posició del gas.

main.py [H] És el mòdul principal del projecte. És l'encarregat de cridar la resta i mostrar les gràfiques resultants. A través d'un petit menú, l'usuari pot seleccionar quina gràfica concreta vol veure o si les vol mostrar totes de cop.

Per tal d'obtenir les gràfiques amb la informació extreta dels diferents sensors, només cal obrir una terminal a la carpeta on hi hagin els fitxers i executar el mòdul `main.py` passant com a paràmetre la ruta als dos arxius de text amb les dades:

```
$python main.py sensors.txt gps.txt
```

El menú que es presenta és senzill, però intuïtiu. Es mostren unes línies d'informació i els punts de menú necessaris per extreure'n les dades.

```
=========================
   Treball Final de Grau
Enginyeria de Sistemes TIC
=========================
Autor: Pol Rodoreda Valeri
Data: 01/06/2016
-------------------------

1. Gràfica inclinació
2. Gràfica GPS
3. Gràfica sensors analògics
4. Totes les gràfiques
5. Sortir

Opció:
```

Mentre l'usuari no seleccioni una de les 5 opcions, s'anirà repetint el procés de demanar opció. Un cop s'hagi mostrat una de les opcions, es torna al mateix menú per continuar treballant. Quan es vol sortir del programa, només cal indicar la opció 5 i es farà una neteja de la terminal perquè quedi neta per tornar a cridar el mòdul `main.py`.

Vegem doncs les gràfiques que es mostraran en cas de seleccionar la quarta opció (les gràfiques a continuació no corresponen a una mateixa prova):

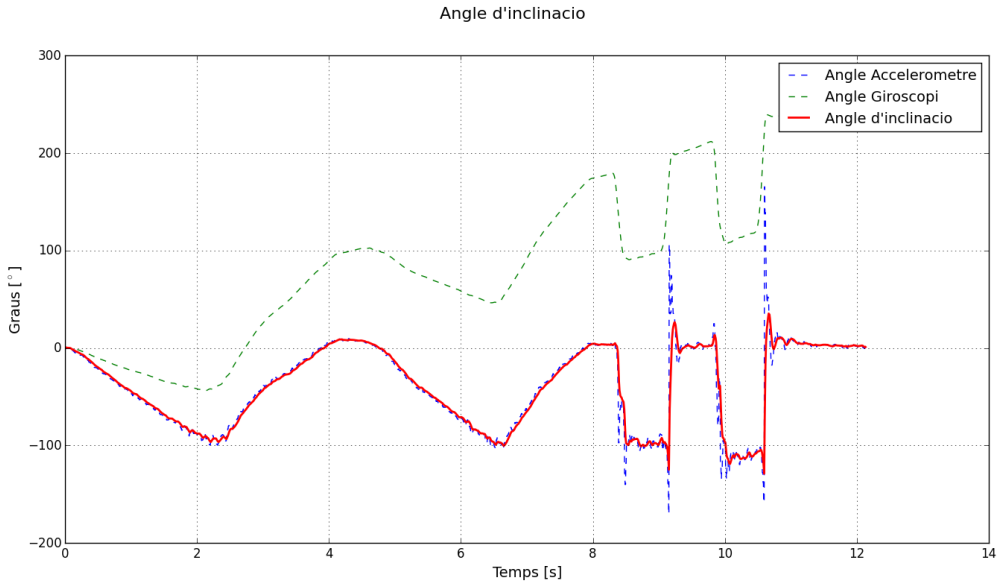


Figura 6.4.: Gràfica angle d'inclinació

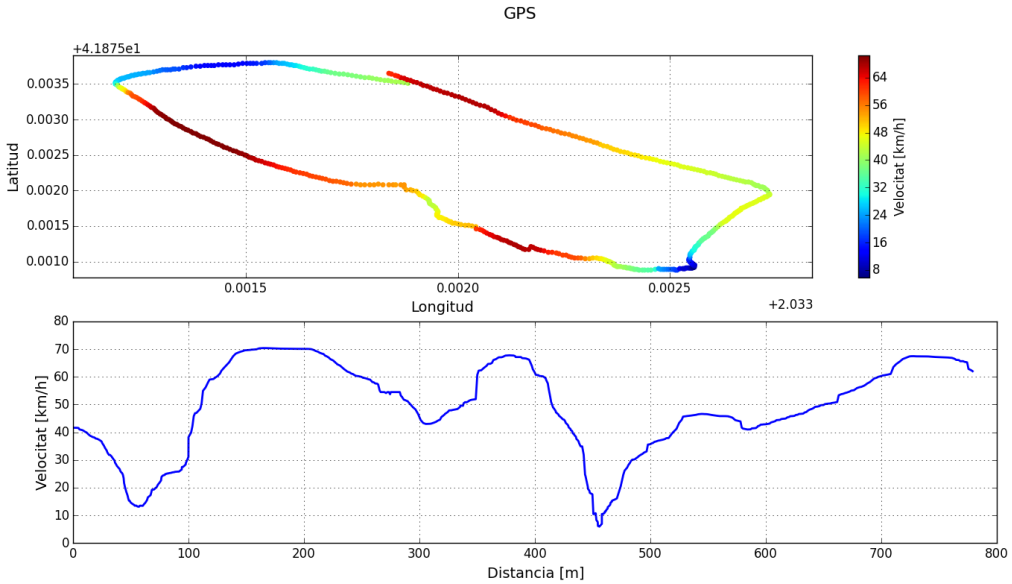


Figura 6.5.: Gràfica GPS

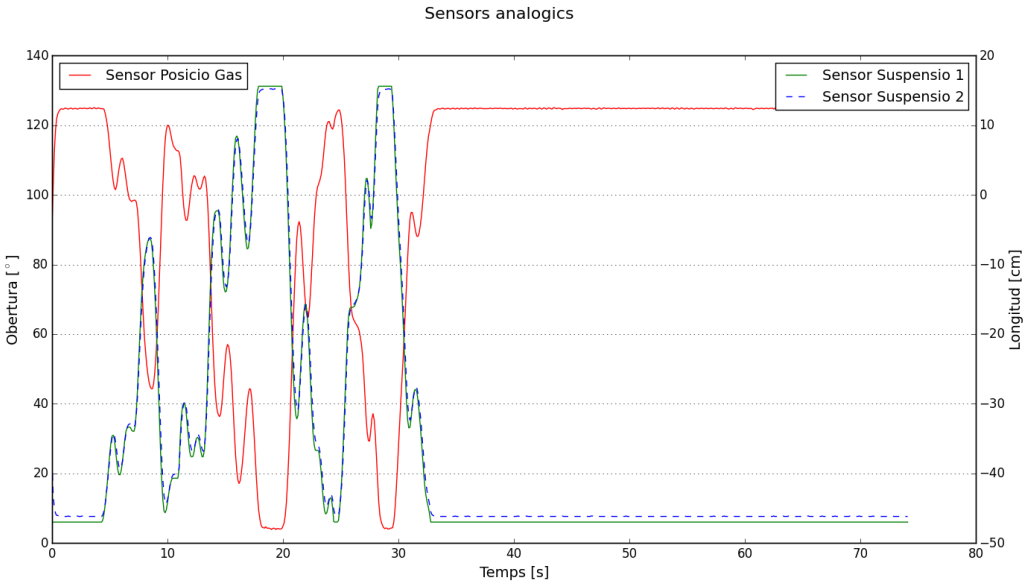


Figura 6.6.: Gràfica sensors analògics

7. Conclusions

Com a valoració final del projecte, es pot dir que el resultat teòric ha estat satisfactori ja que aquest treball ha presentat el procés de disseny i implementació un sistema de monitoratge per a una supermotard de competició, complint els principals objectius als quals havia de fer front.

Primer de tot, aprofundir en el disseny d'un projecte d'aquestes característiques és un acte interessant i enriquidor. Aprendre a lligar els temps d'execució de les diferents parts perquè funcioni tot en harmonia m'ha fet veure que no tot és tan senzill com sembla quan es planteja el problema sobre paper. Tot i que en un primer moment es tinguin les coses clares en ment, la pràctica quasi bé sempre ho desmunta tot. És per aixo que a base d'hores i dedicació, finalment s'ha pogut fer funcionar el sistema tal com s'esperava.

Pel que fa als sensors, m'agradaria pensar que l'alt preu que presenten els sensors de suspensió, es deu a una baixa competència en el sector; aprofitant que el sensor de 75mm no funcionava correctament, es va desmuntar amb la intenció de descobrir, o almenys intuir, què el feia tan car, però la sorpresa va ser que tal com es pensava, es tracta d'un simple potenciòmetre lineal. Si que és veritat que el disseny és elaborat i de qualitat, però la mecànica del funcionament és d'una simplicitat inesperada. Per altra banda la resta de components tenen un preu més baix, el que permet definir el projecte com una solució *low cost* en comparació a la competència de mercat.

Un dels aspectes més enriquidors d'aquest projecte és la metodologia pel càlcul de la inclinació. És aquí on realment s'hi veuen aplicats coneixements adquirits al llarg del grau: filtres, connexions, protocols de comunicació, tipus de dades, procediments matemàtics, etc. Una de les coses que ens fa apreciar la complicació d'aquest tema és que a vegades, el què és senzill per a l'ull humà, no ho és gens tecnològicament; per entendre-ho millor, qualsevol persona, amb més o menys encert, sabria determinar l'angle d'inclinació d'una moto a cop d'ull, mentre que ja s'ha demostrat la complexitat del càlcul a nivell d'enginyeria.

Tot i que era una de les il·lusions que va impulsar aquest projecte a tirar endavant, finalment ha estat una llàstima no poder realitzar les proves en un circuit real, però hagués comportat un allargament en l'entrega del treball; temps que personalment no em puc permetre perdre si vull cursar un màster de cara el curs següent. Tot i això, es va contactar amb el Jaume per intentar fer coincidir algun dia d'entrenament a les dates pròximes a l'entrega del projecte, però resulta que el campionat de supermotard comença el dia 1 de maig i degut a l'ocupació d'hores que li suposa la feina de mecànic, l'entrenament queda al marge centrant-se exclusivament en la competició.

Un tema al qual la gent no sol donar molta importància, però que en realitat s'hi destinen moltes hores i molta inversió, és el disseny i construcció del producte final. En aquest cas, s'ha

partit d'un disseny 3D per a poder realitzar posteriorment el prototip final (apartat 5). Aquest és un primer pas per aconseguir un producte funcional, que ens permet fer proves i verificar els resultats, però no és ni molt menys suficient per exportar al món comercial. És llavors quan el producte canvia radicalment: millora del material, ergonomia, connexions, seguretat, etc.

Una de les traves més grans que han sorgit durant el desenvolupament d'aquest projecte ha estat la combinació del *timming* de tots els components. El principal problema, que no ens esperàvem, són es accessos d'obertura i tancament a la targeta SD. Aprendre a mesurar i visualitzar aquests temps a l'oscil·loscopi per, posteriorment, optimitzar la forma de transmissió de les dades i fer els càlculs adequats per arribar a la conclusió que tot això finalment funciona és una experiència tècnica, enriquidora i interessant.

Realitzar aquest projecte amb LaTeX m'ha aportat el descobriment i coneixement d'una eina molt potent per a la redacció de documents i articles científics. Tot i que al principi la redacció de documents amb LaTeX pot semblar molesta, un cop t'hi acostumes els resultats poder ser increïbles.

Com a conclusió final, m'agradaria indicar que aquest projecte abasta totes les àrees de coneixement adquirides al llarg del grau d'enginyeria de sistemes TIC:

- **Informàtica**: Inclou tota la part del software. Programació de l'AVR, els mòduls que permeten el tractament de dades i l'estudi i implantació de les llibreries.
- **Electrònica**: Abasta tota la part dels sensors, connexions i alimentació. Estudi dels registres de configuració de la IMU i els paràmetres d'alimentació i tensió de sortida dels sensors analògics.
- **Telecomunicacions**: Estudi dels diferents protocols de comunicació amb els mòduls i sensors, la optimització de la forma de guardar les dades a la targeta SD i l'estudi de la configuració i transmissió de les dades del GPS.

7.1. Treball Futur

El desenvolupament d'aquest projecte deixa algunes línies futures a seguir:

- Tot i que les dades del projecte són representades en funció del temps, es podria treballar en un algoritme que permetés detectar el pas per volta i crear una interfície gràfica per mostrar les dades de cada volta i poder fer-ne comparacions al moment. Comentar que en certs circuits, aquests incorporen unes bandes magnètiques que podrien ser usades per contemplar aquest pas per volta.
- Una manera de millorar el càlcul de l'angle d'inclinació és l'aplicació d'un filtre de Kalman[1] substituint l'actual filtre complementari. Aquest tipus de filtre és molt complex i podria ser el tema d'un treball final de grau sencer.

[1] Algoritme desenvolupat per Rudolf E. Kalman l'any 1960 que serveix per identificar l'estat ocult (no mesurable) d'un aiatema dinàmic lineal.

- Un error comès en el plantejament del projecte va ser l'ús de l'Arduino IDE. Aquests tipus d'entorns de desenvolupament integrat, són ideals a nivell amateur. Quan el projecte comença a necessitar varis perifèrics que necessitin un sincronisme ràpid entre ells, és molt recomanable utilitzar el llenguatge de programació C[2]. Per tant, la reescriptura del codi de programació de l'Arduino en C és un treball molt interessant que permetrà optimitzar el funcionament i obtenir un major control del sistema.

- El prototip actual està dissenyat i construït a la mida dels components de *hardware* que s'han utilitzat en aquest projecte (secció 2.2). Per tal de reduir la mida de l'encapsulat i millorar l'eficiència entre les connexions, seria ideal realitzar el disseny d'una placa de circuit imprès. Amb això evitaríem, entre d'altres, desconnexions a causa de les vibracions.

[2]Llenguatge de programació originalment desenvolupat per Dennis M. Ritchie entre 1969 i 1972. És un llenguatge orientat a sistemes operatius, concretament Unix

Bibliografia

[2dd16a] 2d-datarecording. *Linear potentiometer double track*. 2016. URL: http://2d-datarecording.com/Downloads/Datasheets/Sensors-length/Pdf/SA-LPxxxD-000-DINA4.pdf.

[2dd16b] 2d-datarecording. *Moto2*. 2016. URL: http://2d-datarecording.com/en/produkte/custom-packs-1/moto2-custom-pack/.

[5He16a] 5Hertz. *ABC del acelerómetro*. 2016. URL: http://5hertz.com/tutoriales/?p=228.

[5He16b] 5Hertz. *Introducción al giroscopio*. 2016. URL: http://5hertz.com/tutoriales/?p=431.

[Ada16] Lady Ada. *Learn Adafruit Ultimate GPS*. 2016. URL: https://learn.adafruit.com/adafruit-ultimate-gps/overview.

[Álv16] José Antonio E. García Álvarez. *Así funciona la consversión analógico digital*. 2016. URL: http://www.asifunciona.com/electronica/af_conv_ad/conv_ad_5.htm.

[Ard16] Arduino. *Arduino*. 2016. URL: https://www.arduino.cc.

[Aut16] Autodaewoospark. *Sensor TPS (Throttle Position Sensor)*. 2016. URL: http://autodaewoospark.com/sensor-TPS.php.

[Cha16] Efstathios Chatzikyriakidis. *QueueArray library for Arduino*. 2016. URL: http://playground.arduino.cc/Code/QueueArray.

[Dam16] Pablo Damaso. *Guía tarjetas SD y micro SD*. 2016. URL: http://tarjetasd.com/guia/.

[Dev16] Analog Devices. *Digital Accelerometer ADXL345*. 2016. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf.

[Ele16] Electroensaimada. *Tutorial Arduino : I2C*. 2016. URL: http://www.electroensaimada.com/i2c.html.

[Enr16] Enrique. *Como Leer Y Escribir Datos En La Tarjeta SD De Arduino*. 2016. URL: http://www.educachip.com/como-leer-y-escribir-datos-en-la-tarjeta-sd-de-arduino/.

[Fou16] Python Software Foundation. *Python*. 2016. URL: https://www.python.org.

[Inv16] Inc. InvenSense. *ITG3200 Product Specification*. 2016. URL: https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf.

[Lla16] Luis Llamas. *Comunicación de Arduino con puerto serie*. 2016. URL: http://www.luisllamas.es/2014/04/arduino-puerto-serie/.

[Pro16] Prometec. *El bus SPI*. 2016. URL: http://www.prometec.net/bus-spi/.

[Wen16] Kong Wai Weng. *Tutorials by Cytron Technologies. Measuring Tilt Angle with Gyro and Accelerometer*. 2016. URL: http://tutorial.cytron.com.my/2012/01/10/measuring-tilt-angle-with-gyro-and-accelerometer/.

[Wik16a] Wikipedia. *Fórmula del Haversine*. 2016. URL: https://es.wikipedia.org/wiki/F%C3%83%C2%B3rmula_del_Haversine.

[Wik16b] Wikipedia. *Global Positioning System*. 2016. URL: https://en.wikipedia.org/wiki/Global_Positioning_System.

Part II.

Annexes

Annexes del document

Aquest apartat exposa tota la documentació que s'ha generat per desenvolupar aquest Treball Final de Grau.

A. arduino.ino

```
/*-----------------------------------------------------
 * Author: Pol Rodoreda
 * Date: 06/06/2016
 * Last Update: 09/06/2016
 * Version: 1.0
 *
 * File: arduino.ino
 *-----------------------------------------------------*/

#include <TimerOne.h>
#include <QueueArray.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>

/********** DECLARACIO DE VARIABLES GLOBALS **********/
//CUA
QueueArray <byte> queue;
uint8_t item, item2;

//INTERRUPCIONS
#define t_int_us 8000 //Temps entre interrupcions [us]
unsigned long time2 = 0, time1 = 0;

//IMU
#define accel 0x53
int data_registerA = 0x32;
byte valuesA[6];

#define gyro 0x68
int data_registerG = 0x1D;
byte valuesG[6];

//SENSORS
uint8_t tps, sps1, sps2;
```

```
//SD
const int chipSelect = 4;
File file;
File fileGPS;

//GPS
#define RMCONLY "$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"
#define UPDATE_1Hz "$PMTK220,1000*1F\r\n"
#define UPDATE_10Hz "$PMTK220,100*2F\r\n"

/*************** DECLARACIO DE FUNCIONS **************/
//INTERRUPCIONS
void intr_init() {
  Timer1.initialize(t_int_us);
  Timer1.attachInterrupt(timerIsr); //Funcio a cridar quant salti la interrupcio
}

//IMU
void accel_init() {
  //Funció que inicia l'accelerometre configurant els registres adequats
  writeTo(accel, 0x2D, 0x08);
  writeTo(accel, 0x31, 0x08);
}

void gyro_init() {
  //Funció que inicia el giroscopi configurant els registres adequats
  writeTo(gyro, 0x3E, 0x00);
  writeTo(gyro, 0x16, 0x18);
  writeTo(gyro, 0x15, 0x07);
}

//I2C
void writeTo(int device, byte address, byte val) {
  // Funcio per escriure el valor VAL a l'adreça ADDRESS del dispositiu DEVICE
  Wire.beginTransmission(device);
  Wire.write(address);
  Wire.write(val);
  Wire.endTransmission();
}

void readFrom(int device, byte address, int num, byte buff[]){
  //Funcio que ens permet llegir num bytes de l'adreça ADDRESS
  //i guardar-los al buffer BUFF
  Wire.beginTransmission(device);
  Wire.write(address);
  Wire.endTransmission();
```

```
  Wire.beginTransmission(device);
  Wire.requestFrom(device, num); //Demana num bytes del dispositiu esclau device

  int i = 0;
  while (Wire.available()) {
    buff[i] = Wire.read(); //Lectura d'un byte
    i++;
  }
  Wire.endTransmission();
}

//SD
void sd_init() {
  pinMode(10, OUTPUT);
  if (!SD.begin(chipSelect)) {
    return;
  }
  if (SD.exists("sensors.txt") || SD.exists("gps.txt")) {
    SD.remove("sensors.txt");
    SD.remove("gps.txt");
  }
}

//GPS
void gps_init() {
  while (!Serial.available()){
  }
  Serial.write(RMCONLY);
  Serial.write(UPDATE_10Hz);
  Serial.flush();
}

void setup() {
  Serial.begin(9600); //Inicia la comunicacio serie
  Serial.flush();
  delay(500);
  Wire.begin(); //Inicia el bus i2c
  delay(1);
  Wire.setClock(400000); //Velocitat de la comunicacio i2c - 400kHz
  accel_init();
  gyro_init();
  sd_init();
  gps_init();
  intr_init(); //Si configurem abans, comencen a saltar interrupcions i no ens interessa
}
```

```
void loop() {
  //GPS
  if (Serial.available()) {
    fileGPS = SD.open("gps.txt", O_CREAT | O_WRITE);
    if (fileGPS) {
      while (Serial.available()) {
        item = Serial.read();
        fileGPS.write(item);
      }
    }
    fileGPS.close();
  }
  //SENSORS
  if (!queue.isEmpty()) {
    file = SD.open("sensors.txt", O_CREAT | O_WRITE);
    if (file) {
      while (!queue.isEmpty()) {
        item2 = queue.dequeue();
        file.print(item2);
        file.print(",");
      }
    }
    file.close();
  }
}

void timerIsr() {
  interrupts(); //Habilitem interrupcions
  tps = analogRead(A0) >> 2;
  queue.enqueue(tps);
  sps1 = analogRead(A1);
  //DADES ACCELEROMETRE
  readFrom(accel, data_registerA, 6, valuesA);
  queue.enqueue(valuesA[0]);
  queue.enqueue(valuesA[1]);
  queue.enqueue(valuesA[2]);
  queue.enqueue(valuesA[3]);
  queue.enqueue(valuesA[4]);
  queue.enqueue(valuesA[5]);

  sps1 = analogRead(A1) >> 2;
  queue.enqueue(sps1);
  sps2 = analogRead(A2);

  //DADES GIROSCOPI
  readFrom(gyro, data_registerG, 6, valuesG);
  queue.enqueue(valuesG[0]);
```

```
  queue.enqueue(valuesG[1]);
  queue.enqueue(valuesG[2]);
  queue.enqueue(valuesG[3]);
  queue.enqueue(valuesG[4]);
  queue.enqueue(valuesG[5]);

  sps2 = analogRead(A2) >> 2;
  queue.enqueue(sps2);
  tps = analogRead(A0);
}
```

B. getData.py

```
#-*- encoding:utf-8 -*-

##########################################################################
# Author: Pol Rodoreda
# Date: 16/05/2016
# Last Update: 09/06/2016
# Version: 1.0
#
# Script python que llegeix un fitxer de dades separades per comes
# i crea un diccionari de llistes amb les diferents dades segons
# IMU, sensors i GPS.
#
# File: getData.py
##########################################################################

def readFile(gpsFile, sensFile):
    """
    Funcio que transforma el fitxer de dades dataFile
    en un diccionari de llistes amb les dades dels
    diferents sensors.
    """
    gps = []
    imu = []
    analog = []
    data = {}

    #Lectura del fitxer gps.txt
    f = open(gpsFile, "rU")
    line = f.readline()
    while (line != ""):
        if (line[0:5] == "$GPRM"): #Evitem sentencies de configuracio PMTK
            gps.append(line)
        else:
            pass
        line = f.readline()
    f.close()

    data ["gps"] = gps

    #Lectura del fitxer sensors.txt
    g = open(sensFile, "r")
    lines = g.read()

    sline = lines.split(",")
    i = 0
    while (i < len(sline)-14):
        #Accelerometre
        xa = int(sline[i+2]) << 8 | int(sline[i+1])
        if (xa > 32678):
            xa -= 65536
```

```
        elif (xa < -32678)
        ya = int(sline[i+4]) << 8 | int(sline[i+3])
        if (ya > 32678):
            ya -= 65536
        za = int(sline[i+6]) << 8 | int(sline[i+5])
        if (za > 32678):
            za -= 65536
        imu.append(xa)
        imu.append(ya)
        imu.append(za)
        #Giroscopi
        xg = int(sline[i+8]) << 8 | int(sline[i+9])
        if (xg > 32678):
            xg -= 65536
        yg = int(sline[i+10]) << 8 | int(sline[i+11])
        if (yg > 32678):
            yg -= 65536
        zg = int(sline[i+12]) << 8 | int(sline[i+13])
        if (zg > 32678):
            zg -= 65536
        imu.append(xg)
        imu.append(yg)
        imu.append(zg)
        #Sensors analogics
        analog.append(sline[i])
        analog.append(sline[i+7])
        analog.append(sline[i+14])

        i += 15
    g.close()

    data["imu"] = imu
    data["analog"] = analog

    return data
```

C. imu.py

```
#-*- encoding:utf-8 -*-

##########################################################################
# Author: Pol Rodoreda
# Date: 16/05/2016
# Last Update: 09/06/2016
# Version: 1.0
#
# Script python que calcula l'offset de l'accelerometre i el
# giroscopi, tracta les dades i en retorna l'angle d'inclinacio
#
# File: imu.py
##########################################################################

import numpy
import math

def getData(dataImu):
    """
    Funcio que donada una llista de valors dataIMU en retorna
    un diccionari de llistes segons els eixos del sensor
    ja aplicada la conversio a g's i graus/s.
    """
    xa = []
    ya = []
    za = []
    xg = []
    yg = []
    zg = []
    d_imu = {}

    #Conversion factor
    convAcc = 256
    convGiro = 14.375

    i = 0
    while (i < len(dataImu)):
        xa.append(float(dataImu[i])/convAcc)
        ya.append(float(dataImu[i+1])/convAcc)
        za.append(float(dataImu[i+2])/convAcc)
        xg.append(float(dataImu[i+3])/convGiro)
        yg.append(float(dataImu[i+4])/convGiro)
        zg.append(float(dataImu[i+5])/convGiro)
        i += 6

    d_imu["xa"] = xa
    d_imu["ya"] = ya
    d_imu["za"] = za
    d_imu["xg"] = xg
    d_imu["yg"] = yg
```

```
    d_imu["zg"] = zg

    return d_imu

def getOffset(d_imu):
    """
    Funcio que donat el diccionari de valors del eixos dels sensors
    calcula l'offset de cada eix i en retorna un diccionari
    de valors.
    """
    d_offset = {}

    d_offset["xa"] = 1 - numpy.mean(d_imu["xa"][:10])
    d_offset["ya"] = 0 - numpy.mean(d_imu["ya"][:10])
    d_offset["za"] = 0 - numpy.mean(d_imu["za"][:10])
    d_offset["xg"] = 0 - numpy.mean(d_imu["xg"][:10])
    d_offset["yg"] = 0 - numpy.mean(d_imu["yg"][:10])
    d_offset["zg"] = 0 - numpy.mean(d_imu["zg"][:10])

    return d_offset

def calcAngle(d_imu, d_offset):
    """
    Funcio que donat el diccionari de valors i el
    diccionari d'offsets, calcula el valor de les
    dades sense offset i en calcula l'angle d'inclinacio.
    """
    #Obtencio de les dades sense offset
    for i, x in enumerate(d_imu["xa"]):
        d_imu["xa"][i] = x + d_offset["xa"]
    for i, x in enumerate(d_imu["ya"]):
        d_imu["ya"][i] = x + d_offset["ya"]
    for i, x in enumerate(d_imu["za"]):
        d_imu["za"][i] = x + d_offset["za"]
    for i, x in enumerate(d_imu["xg"]):
        d_imu["xg"][i] = x + d_offset["xg"]
    for i, x in enumerate(d_imu["yg"]):
        d_imu["yg"][i] = x + d_offset["yg"]
    for i, x in enumerate(d_imu["zg"]):
        d_imu["zg"][i] = x + d_offset["zg"]

    #Sample time
    dt = 8e-3

    #Time Y-axis
    t = []
    i = 0
    while (i < len(d_imu["xa"])):
        t.append(i * dt)
        i += 1

    #Accelerometer angle
    angleAcc = []
```

```
i = 0
while i < len(d_imu["ya"]):
    angleAcc.append(-(math.atan2(d_imu["ya"][i], d_imu["xa"][i]) \
    * 180/math.pi))
    i += 1

#Low pass filter
angleLPF = []
alpha = 0.9
i = 0
while (i < len(angleAcc)):
    if i == 0:
        angleLPF.append(angleAcc[i])
    else:
        angleLPF.append((1 - alpha) * angleAcc[i] + alpha * angleLPF[i-1])
    i += 1

#Integration of Gyroscope
angleGI = []
i = 0
while i < len(d_imu["zg"]):
    angleGI.append(sum(d_imu["zg"][0:i]) * dt)
    i += 1

#Gyroscope
angleGyro = []
i = 0
while i < len(d_imu["zg"]):
    angleGyro.append(d_imu["zg"][i])
    i += 1

#High pass filter
angleHPF = []
alpha = 0.99
i = 0
while i < len(angleGyro):
    if i == 0:
        angleHPF.append(angleGyro[i] * dt)
    else:
        angleHPF.append(alpha * angleHPF[i-1] + ((angleGyro[i] * dt - \
        angleGyro[i-1] * dt)/(2 * (1 + alpha))))
    i += 1

#Complementari filter
a = 0.98
angleFC = []
i = 0
while i < len(d_imu["zg"]):
    if i == 0:
        angleFC.append(a * (angleGyro[i] * dt) + ((1-a) * angleAcc[i]))
    else:
        angleFC.append(a * (angleFC[i-1] + angleGyro[i] * dt) + ((1-a) * \
        angleAcc[i]))
```

```
        i += 1

    d_angle = {}
    d_angle["angleFC"] = angleFC
    d_angle["angleAcc"] = angleAcc
    d_angle["angleGI"] = angleGI
    d_angle["t"] = t

    return d_angle
```

D. gps.py

```
#-*- encoding:utf-8 -*-

##########################################################################
# Author: Pol Rodoreda
# Date: 16/05/2016
# Last Update: 25/05/2016
# Version: 1.0
#
# Script python que transforma la latitud i longitud del GPS en
# graus decimals i calcula la distancia entre punts a partir de la
# funcio de haversine
#
# File: gps.py
##########################################################################

from haversine import *

def getData(dataGps):
    """
    Funcio que donada una llista de valors dadesGPS en retorna
    un diccionari amb les llistes de les dades de latitud, longitud,
    velocitat i distancia.
    """
    lat = []
    lon = []
    v = []
    d_gps = {}

    i = 0
    while (i < len(dataGps)):
        if (dataGps[i].split(",")[2] == "A"):
            if (dataGps[i].split(",")[4] == "N"):
                lat.append(round(float(dataGps[i].split(",")[3][:2]) + \
                float(dataGps[i].split(",")[3][2:])/60, 5))
            else:
                lat.append(-round(float(dataGps[i].split(",")[3][:2]) + \
                float(data[i].split(",")[3][2:])/60, 5))
            if (dataGps[i].split(",")[6] == "E"):
                lon.append(round(float(dataGps[i].split(",")[5][:3]) + \
                float(dataGps[i].split(",")[5][3:])/60, 6))
            else:
                lon.append(-round(float(dataGps[j].split(",")[5][:3]) + \
                float(dataGps[i].split(",")[5][3:])/60, 6))
            v.append(float(dataGps[i].split(",")[7]) * 1.852) #nusos a km/h
        i += 1

    #Calcul de la distancia entre mostres
    d = []
    i = 0
    while (i < (len(lat) - 1)):
```

```
        if i == 0:
            d.append(0 + haversine(lat[i], lon[i], lat[i+1], lon[i+1]))
        else:
            d.append(d[i-1] + haversine(lat[i], lon[i], lat[i+1], lon[i+1]))
        i += 1

    d_gps["lat"] = lat
    d_gps["lon"] = lon
    d_gps["v"] = v
    d_gps["d"] = d

    return d_gps
```

E. haversine.py

```
#-*- encoding:utf-8 -*-

##########################################################################
# Author: Pol Rodoreda
# Date: 28/04/2016
# Last Update: 25/05/2016
# Version: 1.0
#
# Calcul de la distancia entre dos punts sobre la superficie
# terrestre donades dades de longitud i latitud aplicant la
# Formula de Haversine
#
# File: haversine.py
##########################################################################

from math import sqrt, asin, cos, sin, radians

def haversine(lat1, lon1, lat2, lon2):
    """
    Funcio que donades les dades de latitud i longitud
    de dos punts en retorna la distancia que les separa en metres.
    """
    R = 6372.8 #Earth radius in radians

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlat = lat1 - lat2
    dlon = lon2 - lon1

    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    d = 2 * R * asin(sqrt(a))

    return d * 1000 #Return the distance in meters
```

F. analog.py

```
#-*- encoding:utf-8 -*-

##########################################################################
# Author: Pol Rodoreda
# Date: 17/05/2016
# Last Update: 09/05/2016
# Version: 1.0
#
# Script python que transforma les lectures dels sensors analogics
# en graus i centimetres respectivament.
#
# File: analog.py
##########################################################################

def getData(dataAnalog):
    """
    Funcio que donada una llista de dades dadesAnalog en retorna
    un diccionari amb les llistes de dades dels sensors analogics
    convertides a graus i centimetres respectivament.
    """
    tps = []
    sps1 = []
    sps2 = []
    d_analog = {}

    i = 0
    while (i < len(dataAnalog)):
        tps.append(float(dataAnalog[i]))
        sps1.append(float(dataAnalog[i+1]))
        sps2.append(float(dataAnalog[i+2]))
        i += 3

    #Time Y-axis
    dt = 8e-3
    t = []
    i = 0
    while (i < len(tps)):
        t.append(i * dt)
        i += 1

    #Convertion V - graus
    degreesMax = 125
    ktps = 5.0 / 255
    for i,x in enumerate(tps):
        tps[i] = x * ktps * degreesMax / 3.3

    #Convertion V - cm
    Lmax = 18.3
    Lmin = 2.7
    ksps = 5.0 / 255
```

```
    for i,x in enumerate(sps1):
        sps1[i] = Lmax - float(x * ksps) * (Lmax - Lmin)/5 - Lmin
    for i,x in enumerate(sps2):
        sps2[i] = Lmax - float(x * ksps) * (Lmax - Lmin)/5 - Lmin

    d_analog["tps"] = tps
    d_analog["sps1"] = sps1
    d_analog["sps2"] = sps2
    d_analog["t"] = t

    return d_analog
```

G. chart.py

```
#-*- encoding:utf-8 -*-

########################################################################
# Author: Pol Rodoreda
# Date: 17/05/2016
# Last Update: 09/06/2016
# Version: 1.0
#
# Script python que grafica les dades dels diferents sensors.
#
# File: chart.py
########################################################################

import matplotlib.pyplot as plt

def imuChart(angle):
    """
    Funcio que donat un diccionari amb una llista amb l'angle d'inclinacio
    i el vector de temps, en grafica les dades.
    """
    plt.figure()
    plt.suptitle("Angle d'inclinacio", fontsize = 16)
    plt.plot(angle["t"], angle["angleFC"], "r", label="Angle d'inclinacio", \
    linewidth=2)
    plt.xlabel("Temps [s]", fontsize=14)
    plt.ylabel("Graus [$^\circ$]", fontsize=14)
    plt.legend(fontsize = 14)
    plt.grid(True)

def gpsChart(d_gps):
    """
    Funcio que donada un diccionari amb les dades del gps
    en grafica la velocitat sobre la tracada del circuit
    i la velocitat segons la distancia com a subplot.
    """
    plt.figure()
    plt.suptitle("Dades GPS", fontsize = 16)
    m = plt.subplot(211)
    sp = plt.scatter(d_gps["lon"], d_gps["lat"], c = d_gps["v"], \
    vmin = min(d_gps["v"]), vmax = max(d_gps["v"]), marker = "o", s = 25, \
    linewidth = 0)
    cb = plt.colorbar(sp).set_label("Velocitat [km/h]")
    m.set_xlabel("Longitud", fontsize=14)
    m.set_ylabel("Latitud", fontsize=14)
    m.set_xlim(min(d_gps["lon"]) - 0.0001, max(d_gps["lon"]) + 0.0001)
    m.set_ylim(min(d_gps["lat"]) - 0.0001, max(d_gps["lat"]) + 0.0001)
    plt.grid(True)

    g = plt.subplot(212)
    plt.plot(d_gps["d"], d_gps["v"][:-1], "r", label = "Velocitat[km/h]")
```

```
    g.set_xlabel("Distancia [m]", fontsize=14)
    g.set_ylabel("Velocitat [km/h]", fontsize=14)
    plt.legend(fontsize = 14)
    plt.grid(True)

def analogChart(d_analog):
    """
    Funcio que donat un diccionari amb les dades dels senors analogics
    en grafica les dades en una sola figura i dos eixos, un pel TPS i
    l'altre pels sensors de suspensio.
    """
    plt.figure()
    plt.suptitle("Sensors analogics", fontsize = 16)
    ax1 = plt.subplot(111)
    ax1.plot(d_analog["t"], d_analog["tps"], "r", label = "Sensor Posicio Gas")
    ax1.set_xlabel("Temps [s]", fontsize=14)
    ax1.set_ylabel("Obertura [$^\circ$]", fontsize=14)
    ax1.legend(loc=2, fontsize = 14)

    ax2 = ax1.twinx()
    ax2.plot(d_analog["t"], d_analog["sps1"], "g", label = "Sensor Suspensio 1")
    ax2.plot(d_analog["t"], d_analog["sps2"], "b", label = "Sensor Suspensio 2"\
    ,linestyle = "--")
    ax2.set_ylabel("Longitud [cm]", fontsize=14)
    ax2.legend(loc=1, fontsize = 14)
    plt.grid(True)
```

H. main.py

```
#-*- encoding:utf-8 -*-

##########################################################################
# Author: Pol Rodoreda
# Date: 17/05/2016
# Last Update: 06/06/2016
# Version: 1.0
#
# Script Python que governa la resta. Es l'encarregat de gestionar
# tots els moduls.
#
# File: main.py
##########################################################################

import os
import sys
import getData
import imu
import gps
import analog
import chart
import datetime
import matplotlib.pyplot as plt

def info():
    now = datetime.datetime.now()
    print "=========================="
    print "  Treball Final de Grau "
    print "Enginyeria de Sistemes TIC"
    print "=========================="
    print "Autor: Pol Rodoreda Valeri"
    print "Data: " + now.strftime("%d/%m/%Y")
    print "--------------------------"

def menu():
    print "\n\t1. Grafica inclinacio"
    print "\t2. Grafica GPS"
    print "\t3. Grafica sensors analogics"
    print "\t4. Totes les grafiques"
    print "\t5. Sortir"
    opcio = input("\nOpcio:")
    while (opcio > 5 or opcio < 1):
        opcio = input("Opcio:")
    return opcio

if __name__ == "__main__":
    if (len(sys.argv) < 2):
        print "Especifiqueu el fitxer!\n"
        sys.exit()
    gpsFile = sys.argv[1]
```

```
sensFile = sys.argv[2]
#Data collection
data = getData.readFile(gpsFile, sensFile)
#IMU data
d_imu = imu.getData(data["imu"])
d_offset = imu.getOffset(d_imu)
angle = imu.calcAngle(d_imu, d_offset)
#GPS data
d_gps = gps.getData(data["gps"])
#Analog sensors data
d_analog = analog.getData(data["analog"])
#Menu
info()
opcion = menu()
while (opcion != 5):
    #Charts
    if (opcion == 1):
        if (angle == []):
            print "No hi ha dades enregistrades de la IMU!"
        else:
            chart.imuChart(angle)
            plt.show()
    elif (opcion == 2):
        if (d_gps["lon"] == [] or d_gps["lat"] == []):
            print "No hi ha dades enregistrades de GPS!"
        else:
            chart.gpsChart(d_gps)
            plt.show()
    elif (opcion == 3):
        if (d_analog["tps"] == []):
            print "No hi ha dades enregistrades dels sensors analogics!"
        else:
            chart.analogChart(d_analog)
            plt.show()
    elif (opcion == 4):
        if (angle == []):
            print "No hi ha dades enregistrades de la IMU!"
            chart.gpsChart(d_gps)
            chart.analogChart(d_analog)
            plt.show()
        elif (d_gps["lat"] == []):
            print "No hi ha dades enregistrades de GPS!"
            chart.imuChart(angle)
            chart.analogChart(d_analog)
            plt.show()
        elif (d_analog["tps"] == []):
            print "No hi ha dades enregistrades dels sensors analogics!"
            chart.imuChart(angle)
            chart.gpsChart(d_gps)
            plt.show()
        else:
            chart.imuChart(angle)
            chart.gpsChart(d_gps)
```

```
            chart.analogChart(d_analog)
            plt.show()
    opcion = menu()
os.system("clear")
sys.exit(0)
```